C3

```
/*****************************************************************************
                                                                          ******
**
** File Name:   RSLgettlob.c
**
** Copyright (c) 1998,1999 by EMC Corporation.
**
** Purpose:
**       This module contains the RSTSL_GetTopLevelObjects
**       Restore Service Library function.
**       This function is provided to allow retrieval of the
**       top level objects which are restorable for the given client.
**
**
** Compile-Time Options:
**              This section must list any compile time definitions
**              which will affect this header.
**
*****************************************************************************
                                                                       *****/



/* The following provides an RCS id in the binary that can be located
** with the what(1) utility.  The intent is to keep this short.
*/

#ifndef lint
static char RCS_id [] = "$RCSfile$ "
                        "$Revision$ "
                        "$Date$" ;
#endif


/*
 * Feature test switches.
 * Standard defines required to turn on OS features go here.
 *
 * The following is required for code that uses POSIX API's.
 * Remove for non-POSIX, non-portable code.
 */

#define  _POSIX_SOURCE 1


/*
 * System headers.
 */


/*
 * Epoch headers.
 */
#include <eb/eb_port.h>
#include <eb/rb_log.h>


/*
 * Local headers
 */
#include <RSLinterns.h>


/*
 * #defines, structures, typedefs local to this source file
 */
```



```
typedef struct wi_info
{
    struct wi_info *next;
    char *wi_name;
    char *wi_work;
    char *wi_bic;
    char wi_type;
} wi_info;

NEW_SRC_FILE();

/*
 * External declarations
 */
```

```
/*****************************************************************
 * RSTSL_GetTopLevelObjects:
 *
 *  This function is provided to allow retrieval of the
 *  work items which are restorable for the given client.
 *
 *  It is a GOAL of this routine to return all work-items ever backed
 *  up successfully.  Currently, though, it only looks in the config
 *  file for network workitems of the given client.
 *
 *  The cookie must be initialized to INIT_COOKIE on the first call to
 *  routine.
 *      This routine will update the cookie to allow retrieval of more
 *  objects if there are more than "maxEntries".  The cookie will be
 *  returned as DONE_COOKIE when there are no more to retrieve.
 *
 * Parameters:
 *   sourceHost     (I)  - the name of the source host being restored
 *   maxEntries     (I)  - the maximum number of objects to return
 *   topLevObjs     (O)  - ptr to linked list of Top Level Objects
 *   numberEntries (
 *                  O)  - the real number of objects returned in the array
 *   cookie         (IO) - a place holder for the list position
 *                         meaningful to only the internals of the API
 *
 *****************************************************************/

eerrno_ty
RSTSL_GetTopLevelObjects( const char       *sourceHost,
                          const short      maxEntries,
                          struct RSTRPC_tlo_list **topLevObjs,
                          short            *numberEntries,
                          long             *cookie,
                          int                 execMode)
{
    RBC_WORKGROUP *wgp;
    short        index;
    wi_info      *n_node;
    wi_info      *tmp_list;
    short        tmp_count;
    eerrno_ty    result = E_SUCCESS;
    void         *saved_appdata;
    struct RSTRPC_tlo_list       *tloPtr;
    struct RSTRPC_tlo_list       *tloLast;
    struct pluginData            *piPtr;


    /*
     * Note that the following variables are declared static so that
     * values set during the initial call to this routine will be
     * of use on subsequent calls when no explicit set takes place.
     */

    static       short           wi_count;
    static       short           total_count;
    static       long            valid_cookie;
    static      wi_info          *wi_list;
    static      wi_info          *top_wi_list=NULL;
    static       short           pi_count;
                                      /* num tlos left from plugins */
    static       struct RSTRPC_tlo_list *pluginList = NULL;


    /*
     * Check to make sure rcp has the in memory config info.
```



```
     * If the pointer is NULL, return error.
     */

    if (NULL == rcp->rc_config)
    {
        return(EP_RB_RECOVER_BAD_CONTEXT);
    }

    /*
     * Check value of cookie.  If it is the init cookie, then call
     * rb_get_auth_clients() otherwise, feed off of list of names that
     * were generated by the first call.
     */

    if (NULL == cookie)
    {
        return(EP_RB_RECOVER_BAD_COOKIE);
    }

    /*
     * Fill in the recover context source client hostname field
     * If it currently is set to something, free up that memory first.
     */

    if (NULL != rcp->rc_source_client_hostname)
    {
        free(rcp->rc_source_client_hostname);
    }

    if (sourceHost && *sourceHost != 0)
    {
        rcp->rc_source_client_hostname = esl_strdup(sourceHost);
        if (NULL == rcp->rc_source_client_hostname)
        {
            rec_api_log_csm(SUB_CSM_NOMEM, NULL);
            return(EP_RB_RECOVER_NOMEM);
        }
    }
    else
    {
        return EP_RB_RECOVER_BAD_ARGS;
    }

    rbe_log_debug(
            0, "In GetTLO for %s, cookie = %x", sourceHost, *cookie);

    if (INIT_COOKIE == *cookie)
    {
        /*
          * Set initial values since the cookie indicates the INIT
          */

        wi_count = 0;
         pi_count = 0;
         wi_list = NULL;

        /*
         * Check to see if there is already an in-memory static list
          * workitems being used by this function.  If so,
          * since the caller has reset the cookie to INIT without
          * exhasuted the list.
          */
```

```
206 2           while (NULL != top_wi_list)
207 3           {
208 3               if (NULL != top_wi_list->wi_name)
209 4               {
210 4                   free(top_wi_list->wi_name);
211 3               }
212 3               if (NULL != top_wi_list->wi_work)
213 4               {
214 4                   free(top_wi_list->wi_work);
215 3               }
216 3               if (NULL != top_wi_list->wi_bic)
217 4               {
218 4                   free(top_wi_list->wi_bic);
219 3               }
220 3               tmp_list = top_wi_list->next;
221 3               free(top_wi_list);
222 3               top_wi_list = tmp_list;
223 2           }

225 2   /* Free unused top level objects from plugins */
226 2           if (pluginList)
227 3           {
228 3                   RSTSL_FreeRestorableObjectList( pluginList );
229 3                   pluginList = NULL;
230 2           }

232 2           /*
233 2            * Scan all the workitems in all the workgroups
234 2            * We are going to get all of the items that meet the
                                                                    criteria
235 2            * and keep them in static memory.
                                                Further calls will feed off
236 2            * of this list.
237 2            */

239 2           for (wgp = rcp->rc_config->pgrouplist;
240 2                 wgp != NULL ;
241 2                 wgp = wgp->next)
242 3           {
243 3                RBC_WORKITEM *wip;

245 3                for (wip = wgp->pwilist; wip != NULL; wip = wip->next)
246 4                {
247 4                    /* host must match and witype must not be a plug-in's
                                                                        */
248 4                    if ( (0 == strcmp(wip->sysname, sourceHost))
249 4                      && ((execMode == RAW_NETWORK)
250 4                          || ( 0 == rcp->rc_num_plugin_wi_types
251 4                          || NULL == memchr(
                                    rcp->rc_plugin_wi_types, wip->wi_type,
252 4                                          rcp->rc_num_plugin_wi_types )
                                                                ) ) )
253 5                    {
254 5                        n_node = linked_list_new((
                              generic_list_ty **) &wi_list, sizeof(wi_info));
255 6                        if (n_node == NULL) {
256 6                            result = EP_RB_RECOVER_NOMEM;
257 6                            break;
258 5                        }
259 5                        n_node->wi_name = esl_strdup(wip->name);
260 5                        if (NULL == n_node->wi_name)
261 6                        {
262 6                            result = EP_RB_RECOVER_NOMEM;
263 6                            break;
264 5                        }
```



```
266 5                   /*
267 5                    * For striped workitems, the work item list can be
268 5                    * NULL for the stripes > 1 of workitems without a
269 5                    * partitionspec.
270 5                    */
271 5                       if (
                        NULL != wip->list)      /* else wi_work already NULL */
272 6                       {
273 6                           n_node->wi_work = esl_strdup(wip->list);
274 6                           if (NULL == n_node->wi_work)
275 7                           {
276 7                               result = EP_RB_RECOVER_NOMEM;
277 7                               break;
278 6                           }
279 5                       }

281 5                       if (NULL != wip->backup_init)
282 6                       {
283 6                           n_node->wi_bic = esl_strdup(wip->backup_init);
284 6                           if (NULL == n_node->wi_bic)
285 7                           {
286 7                               result = EP_RB_RECOVER_NOMEM;
287 7                               break;
288 6                           }
289 5                       }

291 5                       n_node->wi_type = wip->wi_type;

293 5                       ++wi_count;
294 4                   }
295 3               } /* end inner for(
                                ) scanning workitems within a workgroup */

297 3               if (
                      result != E_SUCCESS)    /* malloc failure, break out */
298 3                   break;

300 2           } /* end outer for() scanning workgroups */

302 2           total_count = 1;  /* Needs to be 1 based */
303 2           valid_cookie = INIT_COOKIE; /* Clean up cookie crumbs */

305 2           /*
306 2            * Save the pointer to the head of the list for freeing later.
307 2            */

309 2         top_wi_list = wi_list;

311 3           if (result != E_SUCCESS) {
312 4                   if (result == EP_RB_RECOVER_NOMEM) {
313 4                           rec_api_log_csm(SUB_CSM_NOMEM, NULL);
314 3                   }
315 3                   /* free wi_list next time in */
316 3                   return result;
317 2           }

319 2           if (execMode != RAW_NETWORK)
320 3             {
321 3           /* save appData pointer in case a tlo is selected already */
322 3           saved_appdata = rcp->appData;

324 3           /* get top level object list(s) from plugin(s) */
325 3           for ( index = 1, piPtr = rcp->piList;
326 3                 NULL != piPtr;
327 3                 index++, piPtr = piPtr->next )
```

```
328 4                 {
329 4                   tloPtr = NULL;
330 4                   tmp_count = 0;
331 4                   rcp->appData = piPtr->appData;
332 4                   result = piPtr->piFuncArray[PIFuncIndexGetTLO]
333 4                                 (
                                    rcp, sourceHost, &tloPtr, &tmp_count );
334 4                   if ( result != E_SUCCESS
335 4                     || (NULL == tloPtr && tmp_count > 0)
336 4                     || (NULL != tloPtr && tmp_count == 0) )
337 5                   { /* error or inconsistent results */
338 5                           rbe_internal_error(
339 5                                   result,
340 5                                   "plugin: %s;
                                       error in GetTopLevelObjects call",
341 5                                   ((struct pluginIDdata *)(
                                              piPtr->idData))->name
342 5                           );

344 5                           result = E_SUCCESS;     /* not fatal */
345 5                           continue;
                                        /* try next plugin anyway */
346 4                   }
347 4                   if (NULL == tloPtr)
348 4                           continue;       /* none from this plug-in */

350 4                   if (NULL == pluginList)
351 4                           pluginList = tloPtr;
352 4                   else
353 4                           tloLast->next = tloPtr;

355 4                   for ( ; tloPtr && tmp_count;
                                        tmp_count--, pi_count++ )
356 5                   {
357 5                           tloPtr->tlo->root.backupApp = index;
358 5                           tloLast = tloPtr;
359 5                           tloPtr = tloPtr->next;
360 4                   }

362 4                   if (
                      tmp_count || tloPtr)        /* inconsistent output */
363 5                   {
364 5                          rbe_internal_error(
365 5                                  0,
366 5                           "plugin: %s;
                                bad output from GetTopLevelObjects call",
367 5                                ((struct pluginIDdata *)(
                                              piPtr->idData))->name
368 5                           );
369 4                   }
370 3           }

372 3           rcp->appData = saved_appdata;
                                        /* restore saved appData ptr */
373 2             }
374 2           else
375 3             {
376 3               pluginList = NULL;
377 3               pi_count = 0;
378 2             }
379 1       }
380 1       else if ((valid_cookie != *cookie) || (DONE_COOKIE == *cookie))
381 2       {
382 2            return(EP_RB_RECOVER_BAD_COOKIE);
383 1       }
```



```
385 1       /*
386 1        * Logic drops through to here regardless of the passed in cookie
387 1        * being the INIT state or a valid key in a subsequent call.
388 1        */

390 1       /*
391 1        * Return a linked list of the number of "restorable objects"
                                                              requested.
392 1        * Stop if we reach the end of the list.
393 1        */

395 1       /* set linked list and current entry pointers to NULL at start */
396 1       *topLevObjs = tloPtr = NULL;

398 1       index = 0;
399 1       while ( (total_count <= wi_count + pi_count)
400 1               && (index < maxEntries)
401 1               && ( (wi_list != NULL) || (pluginList != NULL) ) )
402 2       {
403 2            if (wi_list != NULL)
404 3           {
405 3                tloPtr = linked_list_new( (
                                        generic_list_ty **) topLevObjs,
406 3                                        sizeof(
                                              struct RSTRPC_tlo_list));
407 3                if ( (NULL == tloPtr)
408 3                  || ((tloPtr->tlo = calloc(1,sizeof(
                                        struct RSTRPC_top_level_obj)) )
409 3                      == NULL) )
410 4                {
411 4                /* do we return partial list, or free list & return none ? */
412 4                /* need a local function to free list of restorable objects
                                                              on error */
413 4                        result = EP_RB_RECOVER_NOMEM;
414 4                        break;
415 3                }

417 3                tloPtr->tlo->root.objLevel = RSTRPC_tlo_type;
418 3                tloPtr->tlo->root.objName = esl_strdup(wi_list->wi_name);
419 3                if (NULL == tloPtr->tlo->root.objName)
420 4                {
421 4                        result = EP_RB_RECOVER_NOMEM;
422 4                        break;
423 3                }

425 3                 tloPtr->tlo->root.backupApp = 0;
                                        /* value for network objects */

427 3                 if (
              NULL != wi_list->wi_work) /* may now be null for oldb kicker */
428 4                 {
429 4                     tloPtr->tlo->fileSpec = esl_strdup(wi_list->wi_work);
430 4                     if (NULL == tloPtr->tlo->fileSpec)
431 5                     {
432 5                         result = EP_RB_RECOVER_NOMEM;
433 5                         break;
434 4                     }
435 3                 }

437 3                 if (NULL != wi_list->wi_bic)
438 4                 {
439 4                     tloPtr->tlo->wiBIC = esl_strdup(wi_list->wi_bic);
440 4                     if (NULL == tloPtr->tlo->wiBIC)
441 5                     {
442 5                         result = EP_RB_RECOVER_NOMEM;
443 5                         break;
```

```
444 4                   }
445 3               }

447 3               tloPtr->tlo->wiType = wi_list->wi_type;

449 3               wi_list = wi_list->next;
450 2           }
451 2           else    /* get tlo from plugin list: */
452 3           {
453 3               if (NULL == *topLevObjs)
454 4               {   /* set start of list ptr to plugin tlo's */
455 4                   *topLevObjs = pluginList;
456 3               }
457 3               else
458 4               {   /* link prev entry to this one in plugin list */
459 4                   tloPtr->next = pluginList;
460 3               }
461 3               tloPtr = pluginList;
462 3               pluginList = pluginList->next;
463 3               tloPtr->next = NULL;
464 2           }

466 2           /* set hostname in all top level objects */
467 2           tloPtr->tlo->hostname = esl_strdup( sourceHost );
468 2           if (NULL == tloPtr->tlo->hostname)
469 3           {
470 3               result = EP_RB_RECOVER_NOMEM;
471 3               break;
472 2           }
473 2           ++total_count;
474 2           ++index;
475 1       }

477 1   /* catch all malloc failures here: free list, return failure (
                                                                    nothing) */
478 2       if (result != E_SUCCESS) {
479 2           rec_api_log_csm(SUB_CSM_NOMEM, NULL);
480 2           index = 0;
                                            /* to indicate none returned */
481 2           valid_cookie = INIT_COOKIE;
                                           /* to indicate restart needed */
482 2           total_count = wi_count + pi_count + 1;
                                            /* to cause free of wi lists */
483 2           /* free linked list output */
484 2           if (*topLevObjs != NULL)
485 3           {
486 3                   RSTSL_FreeRestorableObjectList( *topLevObjs );
487 3                   *topLevObjs = NULL;
488 2           }
489 1       }

491 1       *numberEntries = index;

493 1       /*
494 1        * Set the cookie appropriately.
                                Also store it in the static "valid_cookie"
495 1        */

497 1       if (wi_count + pi_count >= total_count)       /* any more TLO's for
                                                                    later? */
498 2       {                                            /* yes: */
499 3           if (
              wi_count >= total_count) {          /*   any more netwk TLOs? */
500 3                   *cookie = (long)wi_list;
                                            /* yes, still have netwk WIs */
501 2           }
```



```
502 3           else {
503 3                   *cookie = (long)pluginList;
                                                    /* no, rest are plugin TLOs */
504 2           }
505 2           valid_cookie = *cookie;
506 1       }
507 1       else
508 2       {
509 2           /*
510 2            * Set the cookie to DONE state and the valid_cookie to match
                                                                        it.
511 2            */

513 2           if (result == E_SUCCESS)
514 3           { /*dont set cookie on errors */
515 3                   *cookie = DONE_COOKIE;
516 3                   valid_cookie = DONE_COOKIE;
517 2           }

519 2           /*
520 2            * Free up memory.  We are done and don't need it anymore.
521 2            * Start from the top of the list.
522 2            */

524 2           while (NULL != top_wi_list)
525 3           {
526 3               if (NULL != top_wi_list->wi_name)
527 4               {
528 4                   free(top_wi_list->wi_name);
529 3               }
530 3               if (NULL != top_wi_list->wi_work)
531 4               {
532 4                   free(top_wi_list->wi_work);
533 3               }
534 3               if (NULL != top_wi_list->wi_bic)
535 4               {
536 4                   free(top_wi_list->wi_bic);
537 3               }

539 3               tmp_list = top_wi_list->next;
540 3               free(top_wi_list);
541 3               top_wi_list = tmp_list;
542 2           }
543 2           if (pluginList)
544 3           {
545 3                   RSTSL_FreeRestorableObjectList( pluginList );
546 3                   pluginList = NULL;
547 2           }
548 1       }

550 1       return( result );
551     }       /* end of RSTSL_GetTopLevelObjects() */
```

```
/*****************************************************************************
 **
 ** File Name:   RSLgetrobs.c
 **
 ** Copyright (c) 1998,1999 by EMC Corporation.
 **
 ** Purpose:
 **
 **       This module contains the RSTSL_GetRestorableObjects Restore Service
 **       Library API.
 **
 ** Table of Contents:
 ** ------------------
 **
 **               RSTSL_GetRestorableObjects
 **
 **       Internal Functions:
 **
 **               GetTLOContents
 **
 ** Compile-Time Options:
 **               This section must list any compile time definitions
 **               which will affect this header.
 **
 *****************************************************************************
 *****/


/* The following provides an RCS id in the binary that can be located
 ** with the what(1) utility.  The intent is to keep this short.
 */

#ifndef lint
static char RCS_id [] = "$RCSfile$ "
                        "$Revision$ "
                        "$Date$" ;
#endif


/*
 * Feature test switches.
 * Standard defines required to turn on OS features go here.
 *
 * The following is required for code that uses POSIX API's.
 * Remove for non-POSIX, non-portable code.
 */

#define  _POSIX_SOURCE 1


/*
 * System headers.
 */


/*
 * Epoch headers.
 */
#include <eb/eb_port.h>
#include <eb/rb_log.h>
#include <ebutil/eb_locking.h>
```



```
/*
 * Local headers
 */
#include <RSLinterns.h>
#include <RSLstartup.h>


/*
 * #defines, structures, typedefs local to this source file
 */


/*
 * External declarations
 */

NEW_SRC_FILE();


/*
 * Local function prototypes
 */

static eerrno_ty GetTLOContents(
                                   struct RSTRPC_top_level_obj *tloObjPtr,
                                      const boolean_ty allowBF,
                                      const long maxEntries,
                                      struct RSTRPC_uro_list **objListPtr,
                                      long *cntEntries,
                                      long *cookie);
```

```
/*****************************************************************
 *  RSTSL_GetRestorableObjects:
 *
 *  This function is provided to allow retrieval of the
 *  child objects which are restorable given the parent object. The
 *  caller specifies the parent object and whether or not
 *  to include bad files.
 *
 *  Performs the 'get' on the asynchronous thread of the Restore
                                                              Engine.
 *
 * Parameters:
 *   parentPtr      (
                I)  - the parent object: pointer to either a top level
 *                      object or container object
 *   objectLevel    (
           I)  - specifies whether parentObject points to top level or
 *                      container object
 *   objects        (
                  O)  - a pre-allocated pointer to return the list of
 *                      objects in
 *   cookie         (IO) - a place holder for the list position
 *   maxEntries     (I)  - the maximum number of objects to return
 *   cntEntries     (
                O)  - the real number of objects returned in the list
 *   allowBF        (I)  - flag whether or not to include bad files
 *
 *****************************************************************/
eerrno_ty  RSTSL_GetRestorableObjects(
                                    restorableObjectPtr       parentPtr,
                                         enum RSTRPC_ObjectLevel
                                                              objectLevel,
                                    struct RSTRPC_uro_list
                                                              **objects,
                                    long              *cookie,
                                    const long        maxEntries,
                                    long              *cntEntries,
                                    const boolean_ty allowBF )
{
    eerrno_ty    result;
    struct RSTRPC_uro_list *uroList;
    struct RSTRPC_top_level_obj *tloPtr
         = (struct RSTRPC_top_level_obj *)parentPtr;
    struct RSTRPC_user_restorable_object *uroPtr
         = (struct RSTRPC_user_restorable_object *)parentPtr;

    /*
     *  Validate the input arguments.
     */

    if ((NULL == parentPtr) || (NULL == objects) || (
                                                    NULL == cntEntries) )
    {
         return(EP_RB_RECOVER_BAD_ARGS);
    }

    if (objectLevel == RSTRPC_tlo_type) {
         if (tloPtr->root.objName == NULL)
         {
                 return(EP_RB_RECOVER_INVAL_OBJNAME);
         }
    }
    else if (objectLevel == RSTRPC_container_type) {
         if (uroPtr->root.objName == NULL)
         {
                 return(EP_RB_RECOVER_INVAL_OBJNAME);
```



```
         }
         if (rcp->rc_backup_app != uroPtr->root.backupApp)
         {   /* input object not from same app as current TLO */
                 return EP_RB_RECOVER_INVALOP;
         }
    }
    else if (objectLevel == RSTRPC_leaf_type) {
         return(EP_RB_RECOVER_INVALOP);
    }
    else {
         return(EP_RB_RECOVER_INVAL_OBJTYPE);
    }

    /*
     * If cookie is NULL of if it indicates DONE_COOKIE, return error.
     */

    if ((cookie == NULL) || (*cookie == DONE_COOKIE))
    {
         return(EP_RB_RECOVER_BAD_COOKIE);
    }

    /*
     * If EDMRST_Initialize() has not been called, or for some reason
     * the recover_context has not been allocated, then bail out
     * right away!
     */

    if (rcp == NULL)
    {
         return(EP_RB_RECOVER_INVALOP);
    }

    /*
     * If EDMRST_GetTopLevelObjects has not been called,
                                                    then this operation
     * is invalid.
     */

    if (rcp->rc_source_client_hostname == NULL)
    {
         return(EP_RB_RECOVER_INVALOP);
    }

    /*
     * Set number of entries returned to be 0 in case we encounter an
     * error before setting it appropriately.
     */

    *cntEntries = 0;

    /*
     *  If the parent object is a top level object,
                                                call GetTLOContents();
     *  if parent object is a container (
                                        directory), call RSTLL_GetDirContents
     *  for network backups,
                                and RSTPI_GetNextLevelObjects for other backup
     *  apps;
           else return error indicating parent object is not a parent.
     */

    if (objectLevel == RSTRPC_tlo_type) {
         result = GetTLOContents( tloPtr,
                                  allowBF,
                                  maxEntries,
```

```
                                        objects,
                                        cntEntries,
                                        cookie );
    }
    else          /* we already know its a container */
    {
         /*
          * Top Level Object must have been established before this func
          * is called to list dir contents.
          */
         if (rcp->rc_top_level_object_name == NULL)
         {
             return EP_RB_RECOVER_INVALOP;
         }

         if (NULL != rcp->currentPIptr)
         {
                result =
                    rcp->currentPIptr->piFuncArray[PIFuncIndexGetNLO]
                                                ( rcp,
                                                  uroPtr,

                                                RSTRPC_container_type,
                                                  objects,
                                                  cookie,
                                                  maxEntries,
                                                  cntEntries,
                                                  allowBF );
         }
         else
         {
                result = RSTLL_GetDirContents( rcp,
                                                uroPtr->root.objName,
                                                allowBF,
                                                maxEntries,
                                                objects,
                                                cntEntries,
                                                cookie );
         }
    }

    /* if successful, mark all objects with proper backup app: */
    if (E_SUCCESS == result)
    {
         for ( uroList = *objects; uroList; uroList = uroList->next )
                uroList->uro->root.backupApp = rcp->rc_backup_app;
    }

    return result;

}        /* RSTSL_GetRestorableObjects */
```



```
/*
 * GetTLOContents()
 *
 * Function Description:
 *       Given the name of the top level object, set up the restore_context
 *       for it.
 *
 * Parameters:
 *       tloObjPtr   - (I) ptr to the work item restorableObject
 *       allowBF             - (I) flag indicating whether or not to include bad files
 *       maxEntries  - (I) max. # of entries that the preallocated buffer can hold
 *       objListPtr  - (O)  - a pre-allocated pointer to return the list of objects in
 *       cntEntries  - (O) ptr to buffer to receive number of entries returned in objBufPtr
 *       cookie      - (I/O) ptr to a long integer whose value is meaningful to only the internals of the API
 *
 * Return code:
 *       E_SUCCESS       - success. (It is defined as 0 in e_errno.h)
 */

static eerrno_ty
GetTLOContents(  struct RSTRPC_top_level_obj *tloPtr,
                 const boolean_ty allowBF,
                 const long maxEntries,
                 struct RSTRPC_uro_list **objListPtr,
                 long *cntEntries,
                 long *cookie)
{
    char *tloNameP = tloPtr->root.objName;
    char *tNameP = tloPtr->templateName;
    char *rcTmpNameP;
    eerrno_ty rc;
    boolean_ty reset = FALSE;
    int  index;


    /*
     * The environment needs to be reset under the following
     * situations:
     * - it's the first time ever that a top level object is selected;
     * - the caller wants to switch to a top level object different from
     *   what the env is set up for currently;
     *
     * The following criteria are used to determine if the caller
     * wants to switch to a different top level object:
     * - The backup application in the restorableObject is
     *   different from what's kept in the restore_context;
     * - the object name specified in the restorableObject is
     *   different from what's kept in the restore_context;
     * - the object name in the restorableObject is the same
     *   as what's in the restore_context, but the template or
     *   the trailset used is different, or the client host is different;
     */
```

```
320 1     if ( (0 != strcmp(
                      tloPtr->hostname, rcp->rc_source_client_hostname ))
321 1       || (rcp->rc_backup_app != tloPtr->root.backupApp) )
322 2     {
323 2          reset = TRUE;    /* switch to a different application! */
324 1     }

326 1     else if ( (rcp->rc_top_level_object_name == NULL)
327 1          || (0 != strcmp(rcp->rc_top_level_object_name, tloNameP)))
328 2     {
329 2          reset = TRUE;    /* switch to a different object */
330 1     }
331 1     else
332 2     {
333 2          /*
334 2           * rc_top_level_object_name is already set and it's the same
335 2           * as the input top level obj name, we need to further check
336 2           * the template. If the template is the same, we must
337 2           * then check the trailset usage. If any of these is
338 2           * different between what's in the input restorableObject
339 2           * and what's in the restore_context, we need to reset
340 2           * the environment.
341 2           */

343 2          if (rcp->rc_template_defaulted)
344 2                  rcTmpNameP = NULL;
345 2          else
346 2                  rcTmpNameP = rcp->rc_template_name;

348 2          if (NULL != tNameP)
349 3          {
350 3              if (NULL == rcTmpNameP)
351 4              {
352 4                  reset = TRUE;
353 3              }
354 3              else if (strcmp(tNameP, rcTmpNameP)) /* diff templ */
355 4              {
356 4                  reset = TRUE;
357 3              }
358 3              else if (rcp->rc_saveset_thread != tloPtr->ssThread)
359 4              {
360 4                  reset = TRUE;
361 3              }
362 2          }
363 2          else if (NULL != rcTmpNameP)
364 3          {
365 3              reset = TRUE;
366 2          }
367 1     }

369 1     if (reset)
370 2     {
371 2          if (*cookie != INIT_COOKIE)
372 3          {
373 3              return(EP_RB_RECOVER_INVALOP);
374 2          }

376 2 /* if last tlo was other than a network backup object,
                                                  let app clean up: */
377 2          if (rcp->rc_top_level_object_name != NULL)
378 3          {
379 3              if (NULL != rcp->currentPIptr)
380 4              {
381 4                  rc =
                rcp->currentPIptr->piFuncArray[PIFuncIndexClearRC]( rcp );
```



```
382 4                  if (rc != E_SUCCESS)
383 5                  {
384 5                          rbe_internal_error( rc,
385 5                                  "plugin: %s error in
                                                ClearRestoreContext call",
386 5                          ( (struct pluginIDdata *)(
                                        rcp->currentPIptr->idData) )->name );
387 4                  }
388 4                  rcp->currentPIptr->appData = rcp->appData;
                                                    /* save its data */
389 3              }
390 3              else
391 4              {
392 4              /* for network backups, just clear the mark context: */
393 4                  reset_marks( rcp );
394 3              }

396 3              free( rcp->rc_top_level_object_name );
397 3              rcp->rc_top_level_object_name = NULL;
398 2          }

400 2          /*
401 2           * If we're switching to a new source host, we must free
402 2           * the space used for the previous source host name string
                                                                before
403 2           * resetting it to the new.
404 2           */

406 2          if (0 != strcmp(
                        tloPtr->hostname, rcp->rc_source_client_hostname))
407 3          {
408 3              if (NULL != rcp->rc_source_client_hostname)
409 4              {
410 4                  free(rcp->rc_source_client_hostname);
411 3              }
412 3              rcp->rc_source_client_hostname = esl_strdup(
                                                    tloPtr->hostname );
413 3              if (NULL == rcp->rc_source_client_hostname)
414 4              {
415 4                  rec_api_log_csm(SUB_CSM_NOMEM, NULL);
416 4                  return(EP_RB_RECOVER_NOMEM);
417 3              }
418 2          }

420 2          if (0 == (rcp->rc_backup_app = tloPtr->root.backupApp))
421 3          { /* new app is network: */
422 3                  rcp->appData = NULL;
423 3                  rcp->currentPIptr = NULL;
424 2          }
425 2          else  /* find app in plugin list,
                                    set its appData and pluginData ptrs */
426 3          {
427 3                  for (index=1, rcp->currentPIptr = rcp->piList;
428 3                       index < rcp->rc_backup_app &&
                                                rcp->currentPIptr->next ;
429 3                       index++
430 3                      )
431 3                          rcp->currentPIptr = rcp->currentPIptr->next;
432 3                  if (index != rcp->rc_backup_app)
433 4                  { /* got null ptr too early */
434 4                          rbe_internal_error( EP_RB_RECOVER_FATALERR,
435 4                                  "plug-in list corrupted"
                                                                );
436 4                          rec_api_log_csm( SUB_CSM_INTERNAL_ERR, NULL );
437 4                          return EP_RB_RECOVER_FATALERR;
438 3                  }
```

```
440 3                   rcp->appData =  rcp->currentPIptr->appData;
441 2           }

443 2       /*
444 2        * If we're switching to a different tlo, we must:
445 2        * unlock any previous work items and free the workitem name
                                                                    string
446 2        */
447 2           if (rcp->rc_workitem_name != NULL)
448 3           {
449 3               if (rcp->rc_have_wilock)
450 4               {
451 4                   eb_unlock_object(
                                EB_OBJECT_WORKITEM, rcp->rc_workitem_name,
452 4                                   EB_UNLOCK_FREE_IT);
453 4                   rcp->rc_have_wilock = 0;
454 3               }
455 3               free(rcp->rc_workitem_name);
456 3               rcp->rc_workitem_name = NULL;
457 2           }

459 2           if (rcp->rc_template_name != NULL)
460 3           {
461 3               free(rcp->rc_template_name);
462 2           }

464 2           if (tNameP != NULL)
465 3           {
466 3               rcp->rc_template_name = esl_strdup( tNameP );
467 3               rcp->rc_template_defaulted = FALSE;
468 3               if (NULL == rcp->rc_template_name)
469 4               {
470 4                   rec_api_log_csm(SUB_CSM_NOMEM, NULL);
471 4                   return(EP_RB_RECOVER_NOMEM);
472 3               }
473 2           }
474 2           else
475 3           {
476 3               rcp->rc_template_name = NULL;
477 3               rcp->rc_template_defaulted = TRUE;
478 2           }

480 2           rcp->rc_saveset_thread = tloPtr->ssThread;

482 2           /* Do this last,
                        to make sure all strdup's succeed: else leave
483 2            * rc_top_level_object_name NULL */
484 2           rcp->rc_top_level_object_name = esl_strdup(tloNameP);
485 2           if (NULL == rcp->rc_top_level_object_name)
486 3           {
487 3               rec_api_log_csm(SUB_CSM_NOMEM, NULL);
488 3               return(EP_RB_RECOVER_NOMEM);
489 2           }

491 2           /*
492 2            * Check if restore is authorized to the specified client
                                                                    system(s)
493 2            * and by the specified user.
494 2            */

496 2           check_source_sys_admin(rcp);

498 2           if (0 != rcp->rc_backup_app)
499 3           {
500 3                   rc =
```



```
                        rcp->currentPIptr->piFuncArray[PIFuncIndexSetTLO]
501 3                                                                   (
                                                                  rcp, tloPtr);
502 2           }
503 2           else
504 3           {
505 3           /* for network backups, init the workitem & mark context: */
506 3                   rc = RSTLL_SetWorkItem( rcp, tloNameP );
507 3                   if (rc == E_SUCCESS)
508 3                           rc = alloc_plane_arrays( rcp );
509 2           }

511 2           if (rc != E_SUCCESS)
512 3           {
513 3               free( rcp->rc_top_level_object_name );
514 3               rcp->rc_top_level_object_name = NULL;
515 3               return(rc);
516 2           }
517 1       }

519 1       if (0 != rcp->rc_backup_app)
520 2       {
521 2           return rcp -> currentPIptr -> piFuncArray[PIFuncIndexGetNLO]
522 2                                                               ( rcp,
523 2                                                                 tloPtr,

524 2                                                          RSTRPC_tlo_type,
525 2                                                               objListPtr,
526 2                                                               cookie,
527 2                                                               maxEntries,
528 2                                                               cntEntries,
529 2                                                               allowBF );
530 1       }
531 1       else
532 2       {
533 2       /* for network backups retrieve content for the root directory.
                                                                          */

535 2           return RSTLL_GetDirContents( rcp,
536 2                                        "/",
537 2                                        allowBF,
538 2                                        maxEntries,
539 2                                        objListPtr,
540 2                                        cntEntries,
541 2                                        cookie );
542 1       }
543   }           /* GetTLOContents */
```

```
/*****************************************************************************
 **
 ** File Name:   RLLfuncs.c
 **
 ** Copyright (c) 1998,1999 by EMC Corporation.
 **
 ** Purpose:
 **
 **       This module contains the Restore Service Library's Legacy Library
 **       function definitions.
 **
 ** Table of Contents:
 ** ------------------
 **
 **               RSTLL_GetDirContents
 **               RSTLL_SetWorkItem
 **               RSTLL_FillRestObj
 **               RSTLL_AddVolumeToList
 **               RSTLL_RemoveVolumeFromList
 **               RSTLL_FreeVolidList
 **
 **       Internal Functions:
 **
 **               ChDir
 **               ConcatFullName
 **
 ** Compile-Time Options:
 **               This section must list any compile time definitions
 **               which will affect this header.
 **
 *****************************************************************************
                                                                      *****/



/* The following provides an RCS id in the binary that can be located
 ** with the what(1) utility.  The intent is to keep this short.
 */

#ifndef lint
static char RCS_id [] = "$RCSfile$ "
                        "$Revision$ "
                        "$Date$" ;
#endif


/*
 * Feature test switches.
 * Standard defines required to turn on OS features go here.
 *
 * The following is required for code that uses POSIX API's.
 * Remove for non-POSIX, non-portable code.
 */

#define  _POSIX_SOURCE 1


/*
 * System headers.
 */


/*
 * Epoch headers.
```



```
 */
#include <eb/eb_port.h>
#include <eb/rb_log.h>
#include <ebutil/eb_locking.h>
#include <elm/librvm.h>


/*
 * Local headers
 */
#include <ebconfig/rbconfig.h>

#include <restore/RSLlegacy.h>


/*
 * #defines, structures, typedefs local to this source file
 */


/*
 * External declarations
 */

NEW_SRC_FILE();


/*
 * Local function prototypes
 */

static eerrno_ty ChDir( restore_context *rcptr, char *dirName );
/* FindRealNode is not being used, but left in place for possible
   future use.
static eerrno_ty FindRealNode( tree_node *tnPtr, char **fname );
*/
static eerrno_ty ConcatFullName( restore_context *rcptr,
                                 tree_node *tnPtr,
                                 char **fname );



/*
 * Function: RSTLL_SetWorkItem()
 *
 * Function Description:
 *       This function sets up the necessary fields in the restore_context
 *       structure according to the specified work item.
 *
 * Parameter:
 *       rcp     (I) pointer to restore context
 *       wiNameP         (I) ptr to name of the work item.
 *
 * Return Codes:
 *       E_SUCCESS       - success
 *       EP_RB_RECOVER_INVALWI   - Invalid work item
 *       EP_RB_RECOVER_MCAT_ERR  - rc_newmcat() all failed
 *
 * Note:
 *       The rc_pwd field in recover_context is set to "/" in rc_newmcat().
 */

eerrno_ty
RSTLL_SetWorkItem( restore_context *rcp, char *wiNameP )
{
```

```
128 1     eerrno_ty errnum;

130 1     /*
131 1      * Clear fields in recover_context that are related to
132 1      * the previously selected workitem/backup.
133 1      * Note that when rc_catdate is zero, it means that there
134 1      * is not a valid current backup.
135 1      */

137 1     if (rcp->rc_have_wilock)    /* free previous workitem lock,
                                                                  if any */
138 2     {
139 2         rbe_log_debug( 0, "RSTLL_SetWorkItem: Unlocking %s ",
140 2                        rcp->rc_workitem_name );

142 2         eb_unlock_object(EB_OBJECT_WORKITEM, rcp->rc_workitem_name,
143 2                          EB_UNLOCK_FREE_IT);
144 2         rcp->rc_have_wilock = 0;
145 1     }

147 1     rcp->rc_catdate = (time_t)0;
148 1     memset(rcp->rc_catdate_str, 0, CATDATE_STRING_SIZE);

150 1     /* Set the current workitem name: */
151 1     rcp->rc_workitem_name = esl_strdup(wiNameP);
152 1     if (NULL == rcp->rc_workitem_name)
153 2     {
154 2          rec_api_log_csm(SUB_CSM_NOMEM, NULL);
155 2          return(EP_RB_RECOVER_NOMEM);
156 1     }

158 1     /*
159 1      * Reset the multicat structures for this work item.
160 1      * rc_newmcat() will lock the work item (indirectly
161 1      * through the routines it calls).
162 1      */

164 1     errnum = rc_newmcat(rcp); /* set up the mcat for the work item */
165 1     if (0 != errnum)
166 2     {
167 2          /*
168 2           * If rc_newmcat() failed, we must unlock the work item
169 2           * and reset the env.
170 2           */

172 2          if (rcp->rc_have_wilock)
173 3          {
174 3              rbe_log_debug( 0, "RSTLL_SetWorkItem: Unlocking %s ",
175 3                             rcp->rc_workitem_name );
176 3              eb_unlock_object(
                                 EB_OBJECT_WORKITEM, rcp->rc_workitem_name,
177 3                               EB_UNLOCK_FREE_IT);
178 3              rcp->rc_have_wilock = 0;
179 2          }

181 2          free(rcp->rc_workitem_name);
182 2          rcp->rc_workitem_name = NULL;
183 2 #if 0     /* I dont think we want to do this for all apps */
184 2          if (NULL != rcp->rc_template_name)
185 3          {
186 3              free(rcp->rc_template_name);
187 3              rcp->rc_template_name = NULL;
188 2          }
189 2          rcp->rc_saveset_thread = EB_THREAD_PRIMARY;
190 2 #endif
191 2          if (
```



```
     EP_RB_CAT_NO_SSELEM != errnum && EP_RB_RECOVER_NO_SAVESET != errnum)
192 3         {
193 3             rbe_internal_error(errnum, "rc_newmcat failed.");
194 2         }
195 2         return(errnum);
196 1     }
197 1     if (NULL == rcp->rc_template_name)
198 2     {
199 2       cat_descriptor *catdPtr;
200 2       rbcat_head_t cathdr;
201 2       rbsaveset_t saveset;

203 2       catdPtr = mcat_getcatd(rcp->rc_mcp, mcat_get_mcplane(
                                                              rcp->rc_mcp));
204 2       if (NULL == catdPtr)
205 3       {
206 3         return(EP_RB_RECOVER_NO_CATALOG);
207 2       }
208 2       if (0 != catd_get_cat_head(catdPtr, &cathdr))
209 3       {
210 3         return(EP_RB_RECOVER_NO_CATALOG);
211 2       }

213 2       if (0 != ss_find1(&cathdr.ch_savesetID, &saveset, 0))
214 3       {
215 3          return(EP_RB_RECOVER_NO_SAVESET);
216 2       }

218 2       rcp->rc_template_name = esl_strdup(saveset.ss_templ);
219 2       if (NULL == rcp->rc_template_name)
220 3       {
221 3          rec_api_log_csm(SUB_CSM_NOMEM, NULL);
222 3          return(EP_RB_RECOVER_NOMEM);
223 2       }
224 2       rcp->rc_saveset_thread = saveset.ss_thread;
225 1     }

227 1     return(E_SUCCESS);
228   }         /* RSTLL_SetWorkItem */
```

```
230  /*
231   * Function: RSTLL_GetDirContents()
232   *
233   * Function Description:
234   *       This function returns the contents of the current directory
                                                                    in the
235   *       preallocated restorableObject buffers.
236   *
237   * Parameters:
238   *      rcp           - (I) Pointer to the restore context
239   *      dirName       - (I) ptr to the directory full pathname string
240   *                          (NULL terminated, "/" for top of workitem)
241   *      allowBF       - (I) flag for whether or not to include bad files
242   *      maxEntries    - (I) max. # of entries to return
243   *      objListPtr    - (
                        O) a pointer to receive the start of the object list
244   *      cntEntries    - (
                      O) ptr to buffer to receive number of entries returned
245   *      cookie        - (
                       I/O) ptr to a token used in successive calls when
246   *                          more objects remain.
                                                    The input value must be
247   *                          INIT_COOKIE on the first call for a work
                                                                  item, and
248   *                          will be DONE_COOKIE on return if the last
                                                                 object for
249   *                          a workitem is returned;
                                                 otherwise it is meaningful
250   *                          only to the internals of the API.
251   *
252   * Return Codes:
253   *       E_SUCCESS                - success
254   *       EP_RB_RECOVER_BAD_ARGS   - invalid input argument
255   *       EP_RB_RECOVER_INVALDIR   - cannot find tree_node for curr dir
256   *       EP_RB_RECOVER_BAD_COOKIE        - invalid cookie
257   *       EP_RB_RECOVER_INVALOP    - invalid operation
258   *       EP_RB_RECOVER_FATALERR   - internal inconsistency
259   *
260   */

262  eerrno_ty
263  RSTLL_GetDirContents( restore_context            *rcp,
264                        char                       *dirName,
265                        const boolean_ty           allowBF,
266                        const long                 maxEntries,
267                        struct RSTRPC_uro_list     **objListPtr,
268                        long                       *cntEntries,
269                        long                       *cookie )
270  1 {
271  1     eerrno_ty    rtc = 0;
272  1     tree_node   *tnp = NULL;
273  1     static tree_node    **dir_ele = NULL;
274  1     tree_node    **tnpp;
275  1     static long valid_cookie = INIT_COOKIE;
276  1     static int  nitems = 0;
277  1     struct RSTRPC_user_restorable_object       *objPtr;
278  1     struct RSTRPC_uro_list                     *listPtr = NULL;
279  1     int output_count;
280  1     rbcat_elem_t cat_elem;


283  1     if (objListPtr == NULL)
284  2     {
285  2          return(EP_RB_RECOVER_BAD_ARGS);
286  1     }
```



```
288  1     /*
289  1      * If it's the first call on the specified directory, we must:
290  1      * 1. do a chdir operation to get the recover_context environment
                                                                    set up
291  1      *    appropriately;
292  1      * 2. free the dir_ele array and reset tree_node count nitems;
293  1      * 3. get all the tree_nodes for the directory contents in dir_ele
294  1      */

296  1     if (*cookie == INIT_COOKIE)
297  2     {
298  2          if ((rtc = ChDir(rcp, dirName)) != E_SUCCESS)
299  3          {
300  3              return(rtc);
301  2          }

303  2          if (NULL != dir_ele)
304  3          {
305  3              mcat_array_rls(rcp->rc_mcp, dir_ele);
306  3              dir_ele = NULL;
307  2          }

309  2          nitems = 0;
310  2          valid_cookie = INIT_COOKIE;

312  2          /*
313  2           * Get the tree_node for the current dir, which is needed for
314  2           * the next mcat call.
315  2           */

317  2          if ((tnp = mcat_get_current_dir(rcp->rc_mcp)) == (
                                                          tree_node *)NULL)
318  3          {
319  3              return(EP_RB_RECOVER_INVALDIR);
320  2          }
321  2          nitems = mcat_array_children(rcp->rc_mcp, tnp, &dir_ele);
322  2          tnpp = dir_ele;
323  1     }
324  1     else
325  2     {
326  2          /*
327  2           * We're doing a follow-up query on a directory.
                                                          If the cookie has
328  2           * value different from where it was left off,
                                                        then this operation
329  2           * is invalid and an error code is returned. If the specified
330  2           * dirName is different from what's set previously,
                                                             then this is an
331  2           * invalid query;
                                otherwise set the tree_node ptr to what's in the
332  2           * cookie.
333  2           */

335  2          if (*cookie != valid_cookie)
336  3          {
337  3              return(EP_RB_RECOVER_BAD_COOKIE);
338  2          }

340  2          if (0 != strcmp(dirName, rcp->rc_pwd))
341  3          {
342  3              return(EP_RB_RECOVER_INVALOP);
343  2          }

345  2          tnpp = (tree_node **)*cookie;
346  1     }
```

```
348 1     if (nitems == 0)
349 2     {
350 2          *cntEntries = 0;
351 2          *cookie = DONE_COOKIE;
352 2          valid_cookie = DONE_COOKIE;

354 2          /*
355 2           * Release the memory when we're done
356 2           */

358 2          if (NULL != dir_ele)
359 3          {
360 3              mcat_array_rls(rcp->rc_mcp, dir_ele);
361 3              dir_ele = NULL;
362 2          }
363 2          return(E_SUCCESS);
364 1     }

366 1     output_count = 0;
367 1     while ((nitems > 0) && (output_count < maxEntries))
368 2     {
369 3          if (*objListPtr == NULL) {
370 3                  *objListPtr = calloc( 1, sizeof(
                                          struct RSTRPC_uro_list ));
371 3                  listPtr = *objListPtr;
372 2          }
373 3          else {
374 3                  listPtr->next = calloc( 1, sizeof(
                                          struct RSTRPC_uro_list ));
375 3                  listPtr = listPtr->next;
376 2          }
377 3          if (NULL == listPtr) {  /* test for malloc failure */
378 3                  rtc = EP_RB_RECOVER_NOMEM;
379 3                  break;
380 2          }
381 2          objPtr = calloc( 1, sizeof(
                              struct RSTRPC_user_restorable_object ));
382 3          if ( (listPtr->uro = objPtr) == NULL) {
383 3                  rtc = EP_RB_RECOVER_NOMEM;
384 3                  break;
385 2          }

387 2          tnp = *tnpp;
388 2          if (tnp == NULL)
389 3          {
390 3                  rtc = EP_RB_RECOVER_FATALERR;
391 3                  break;
392 2          }

394 2          /*
395 2           * get the catalog element
396 2           */

398 2          mcat_getcatlm(rcp->rc_mcp, tnp, &cat_elem);

400 2          /*
401 2           * Check if the caller wants to include bad files and
402 2           * the status of the current object in these areas
403 2           */

405 2          if (!allowBF && (cat_elem.ce_status & CESTAT_BADDATA))
406 3          {
407 3              tnpp++;
408 3              nitems--;
409 3              continue;
```



```
410 2          }

412 2          /* fill output buffer */
413 2          rtc = RSTLL_FillRestObj(rcp, &cat_elem, tnp, objPtr);
414 2          if (
          rtc != 0)   /* if something went wrong in RSTLL_FillRestObj(), */
415 3          {
416 3              break;      /* we must stop processing further */
417 2          }
418 2          output_count++;
419 2          tnpp++;
420 2          nitems--;
421 1     }

423 1     *cntEntries = output_count;

425 1     /*
426 1      * If we exited the loop with the prealloc'd buffers filled,
                                                          we may or
427 1      * may not have exhausted the dir contents
428 1      */

430 1     if (output_count == maxEntries)
431 2     {
432 2          if (
              nitems == 0)                /* dir contents exhausted,      */
433 3          {
434 3              *cookie = DONE_COOKIE;
                                          /* set the cookie accordingly   */
435 2          }
436 2          else                              /* otherwise,
                                                     store the tree    */
437 3          {
438 3              *cookie = (long) tnpp;
                                          /* node ptr in the cookie       */
439 2          }
440 1     }
441 1     else
442 2     {
443 2          /*
444 2           * If we exited the loop without filling the provided buffers,
445 2           * then we must have exhausted the dir contents
446 2           */

448 2          *cookie = DONE_COOKIE;
449 1     }

451 1     /*
452 1      * Release the memory when we're done
453 1      */

455 1     if (*cookie == DONE_COOKIE)
456 2     {
457 2          if (NULL != dir_ele)
458 3          {
459 3              mcat_array_rls(rcp->rc_mcp, dir_ele);
460 3              dir_ele = NULL;
461 2          }
462 2          valid_cookie = DONE_COOKIE;
463 1     }
464 1     else
465 2     {
466 2          valid_cookie = *cookie;
467 1     }

469 1     if (rtc != 0)
```

```
470 2       {
471 2            /* free memory allocated here */
472 2            RSTSL_FreeRestorableObjectList( objListPtr );
473 2           *objListPtr = NULL;       /* leave NULL in *objListPtr */

475 2            return(rtc);
476 1       }
477 1       else
478 2       {
479 2            return(E_SUCCESS);
480 1       }
481     }         /* RSTLL_GetDirContents */
```



```
483     /*
484      * Function: RSTLL_FillRestObj()
485      *
486      * Function Description:
487      *       This function fills most of the fields of the
                                                        restorableObject for
488      *       a file or directory object using info contained in the input
                                                              catalog
489      *       element structure and tree_node structure.
490      *
491      * Parameters:
492      *       rcp                       - (I) Pointer to the restore context
493      *       catElemPtr                - (
                                             I) ptr to catalog element structure
494      *       tnPtr                     - (I) ptr to tree_node
495      *       objPtr                    - (
                                   O) ptr to the RSTRPC_user_restorable_object
496      *                                   to be filled
497      * Return Codes:
498      *       E_SUCCESS                 - operation is successful;
499      *       EP_RB_RECOVER_FATALERR    - inconsistency exists in the recover
500      *                                   internals, caller should abort;
501      *      EP_RB_RECOVER_NOMEM      - not enough memory
502      *
503      * Note:
504      *       The portion of code that calculates the size of the directory
                                                                        is
505      *       copied from sizestring() in src/server/libs/ebrecover/
506      *       grandfathered/cmd_ls.c
507      */

509     eerrno_ty
510     RSTLL_FillRestObj(restore_context *rcp,
511                  rbcat_elem_t *catElemPtr,
512                  tree_node *tnPtr,
513                  struct RSTRPC_user_restorable_object *objPtr)
514 1   {
515 1       /*
516 1        * The rbcat_elem_t passed in as the first arg does not contain
517 1        * the object's full pathname - this is because mcat_getcatlm()
518 1        * does not call catd_read_catlm() with a preallocated buffer.
519 1        * Call tlmno2recs() which will fill the pathname if everything
520 1        * in the catalog was correct.
521 1        */

523 1       char *fname;
524 1       char *tmp_uid;
525 1       char *tmp_gid;
526 1       cat_descriptor *catd;
527 1       tree_node **tn_array;
528 1       int nchildren;
529 1       u_long phony_size;
530 1       rbtree_elem_t tree_elem;
531 1       int errnum;
532 1       eerrno_ty rtc;
533 1       netBackupObjData *appDataPtr;

535 1       /* validate inputs: */
536 1       if (NULL == rcp || NULL == catElemPtr || NULL == tnPtr || NULL ==
                                                                    objPtr)
537 2       {
538 2            rec_api_log_csm(SUB_CSM_INTERNAL_ERR, NULL);
539 2           return EP_RB_RECOVER_FATALERR;
540 1       }
```

```
542 1          /* alloc and clear net backup specific (opaque) structure */
543 1      appDataPtr = calloc( 1, sizeof(netBackupObjData) );
544 1      if (NULL == appDataPtr)
545 2      {
546 2           rec_api_log_csm(SUB_CSM_NOMEM, NULL);
547 2           return(EP_RB_RECOVER_NOMEM);
548 1      }
549 1      objPtr->appData.length = sizeof(netBackupObjData);
550 1      objPtr->appData.data = (char *)appDataPtr;

552 1      objPtr->root.backupApp = rcp->rc_backup_app;
                                                /* might not be network */

554 1      errnum = tlmno2recs(rcp,
555 1                          tnPtr->tn_index,
556 1                          tnPtr->tn_mcplane,
557 1                          &tree_elem,
558 1                          catElemPtr,
559 1                          &fname);

561 1      if ((errnum != 0) || (fname == NULL))
562 2      {
563 2           /*
564 2            * When tlmno2recs() returns an error and a NULL name ptr,
565 2            * it's most likely that we're dealing with a tree_node that
566 2            * does not have a valid corresponding catalog element,
567 2            * therefore we cannot find the node's full pathname. This
568 2            * happens when the work item has filespec such as "DO_DIR
569 2            * /x/y/z". ConcatFullName() concatenates the full pathname
570 2            * of the object by traversing the tree_node chain upwards.
571 2            */

573 2           rtc = ConcatFullName(rcp, tnPtr, &fname);
574 2           if (E_SUCCESS != rtc)
575 3           {
576 3               return(rtc);
577 2           }

579 2           /*
580 2            * ConcatFullName has malloc'd the fullname string buffer,
581 2            * so we don't need to reallocate it.
582 2            */

584 2           objPtr->root.objName = fname;
585 1      }
586 1      else
587 2      {
588 2           objPtr->root.objName = esl_strdup(fname);
589 1      }

591 1      if (NULL == objPtr->root.objName)
592 2      {
593 2           rec_api_log_csm(SUB_CSM_NOMEM, NULL);
594 2           return(EP_RB_RECOVER_NOMEM);
595 1      }

597 1      appDataPtr->objSSID = catElemPtr->ce_ssID;
598 1      appDataPtr->objBFID = catElemPtr->ce_bitfileID;

600 1      objPtr->objMode = catElemPtr->ce_mode;

602 1      if (   (NULL == (tmp_uid = uid2str((int)catElemPtr->ce_owner)))
603 1          || (NULL == (tmp_gid = gid2str((int)catElemPtr->ce_group))))
604 2      {
605 2          return EP_RB_RECOVER_NOMEM;
606 1      }
```



```
608 1      objPtr->objOwnerName = esl_strdup(tmp_uid);
609 1      if (NULL == objPtr->objOwnerName)
610 2      {
611 2           rec_api_log_csm(SUB_CSM_NOMEM, NULL);
612 2           return(EP_RB_RECOVER_NOMEM);
613 1      }

615 1      objPtr->objGroupName = esl_strdup(tmp_gid);
616 1      if (NULL == objPtr->objGroupName)
617 2      {
618 2           rec_api_log_csm(SUB_CSM_NOMEM, NULL);
619 2           return(EP_RB_RECOVER_NOMEM);
620 1      }

622 1      objPtr->objModTime = catElemPtr->ce_mtime;
623 1      if (catElemPtr->ce_status & CESTAT_BADDATA)
624 2      {
625 2           objPtr->objBackupStatus = RSTRPC_Backup_Bad;
626 1      }
627 1      else
628 2      {
629 2           objPtr->objBackupStatus = RSTRPC_Backup_Good;
630 1      }
631 1      appDataPtr->objtnMCplane = tnPtr->tn_mcplane;
632 1      appDataPtr->objtnIndex = tnPtr->tn_index;

634 1      if ((NULL == tnPtr->tn_name) && (tnPtr->tn_flags & TNF_ROOTDIR))
635 2      {
636 2         objPtr->objBaseName = esl_strdup("/");
637 1      }
638 1      else
639 2      {
640 2         objPtr->objBaseName = esl_strdup(tnPtr->tn_name);
641 1      }

643 1      if (NULL == objPtr->objBaseName)
644 2      {
645 2           rec_api_log_csm(SUB_CSM_NOMEM, NULL);
646 2           return(EP_RB_RECOVER_NOMEM);
647 1      }

649 1      /*
650 1       * Calculation of the size of the object depends on the type of
                                                                        the
651 1       * object as follows:
652 1       *           if the object is a directory, we estimate its size;
653 1       *           if the object is a file,
                                                we get its true size in bytes;
654 1       *           if the object is a block or char device node,
                                                                    we treat
655 1       *           it as a file but with size 0.
656 1       */

658 1      if (S_ISDIR(catElemPtr->ce_mode))
659 2      {
660 2          /*
661 2           * Directory sizes are not stored in the catalogs.
662 2           * We construct a phony size based on the number
663 2           * of children in the directory, just so that
664 2           * the size in the ls listing looks "comforting",
665 2           * and to help suggest which directories have lots
666 2           * of files vs.  few files.
667 2           */

669 2           objPtr->root.objLevel = RSTRPC_container_type;
```

```
671  2          /*
672  2           * do this just to get the number of children
673  2           */

675  2          nchildren = mcat_array_children(
                                        rcp->rc_mcp, tnPtr, &tn_array);
676  2          mcat_array_rls(rcp->rc_mcp, tn_array);

678  2          /*
679  2           * derive phony size @ 20bytes/entry, rounded to 512 mult
680  2           */

682  2          phony_size = nchildren * 20;
683  2          phony_size = (phony_size + 511) / 512;
684  2          phony_size = phony_size * 512;
685  2          if (phony_size == 0)
686  3          {
687  3              phony_size = 512;
688  2          }

690  2          objPtr->objSize.low = phony_size;
691  2          objPtr->objSize.high = 0;
692  1      }
693  1      else
694  2      {
695  2          objPtr->root.objLevel = RSTRPC_leaf_type;

697  2          /*
698  2           * For char or block device node files, the size is used to
699  2           * indicate its devno.
700  2           */

702  2          if (S_ISCHR(catElemPtr->ce_mode) || S_ISBLK(
                                                catElemPtr->ce_mode))
703  3          {
704  3              objPtr->objSize.low = catElemPtr->ce_rdev;
705  3              objPtr->objSize.high = 0;
706  2          }
707  2          else
708  3          {
709  3              objPtr->objSize.high = catElemPtr->ce_filesize.high;
710  3              objPtr->objSize.low = catElemPtr->ce_filesize.low;
711  2          }
712  1      }

714  1      return(E_SUCCESS);
715    }         /* RSTLL_FillRestObj */
```



```
717  /*
718   * Function: ChDir()
719   *
720   * Function Description:
721   *       This function changes the current directory to the specified
                                                                  directory.
722   *       All the fields in the recover_context structure related to
                                                                the current
723   *       directory get reset appropriately.
724   *
725   * Parameters:
726   *      rcptr    - (I) Pointer to the restore context
727   *       dirName          - (
           I) ptr to the directory full pathname NULL-terminated string
728   *
729   * Return Codes:
730   *       E_SUCCESS                - success
731   *       EP_RB_RECOVER_INVALDIR  - invalid directory
732   *       EP_RB_RECOVER_PERMISSION_DENIED          - caller has no
                                                              access rights
733   *       EP_RB_RECOVER_MCAT_ERR  - multicat error
734   *
735   * Note:
736   *       This function was copied from the original xebrecover's
                                                                 base_sys.c:
737   *       rb_change_dir(
                          ), which is an existing working piece, with minor
738   *       modifications. To reduce risk, the original program logic is
739   *       preserved.
740   */

742  static eerrno_ty
743  ChDir( restore_context *rcptr, char *dirName)
744  1 {
745  1      tree_node   *ntnp;
746  1      int     err = 0;
747  1      char    *save_current_pwd;


750  1      save_current_pwd = rcptr->rc_pwd;    /* May be freed later */

752  1      ntnp = mcat_lookup_path_chk(rcptr->rc_mcp,
753  1                                  dirName,
754  1                                  rcptr->rc_effective_uid,
755  1                                  rcptr->rc_effective_gids,
756  1                                  rcptr->rc_effective_ngids,
757  1                                  (char **)NULL,
758  1                                  &err);
759  1      /*
760  1       * mcat_lookup_path_chk(
                               ) returns 0 for success or one of three error
761  1       * codes if failed: EACCES, ENOTDIR,
                                     and ENOENT. In the case of success,
762  1       * however,
                    it's possible that no tree_node for the specified dirName
763  1       * was found, i.e. ntnp == NULL. If ntnp is NULL,
                                                        or if it indicates
764  1       * that the tree_node is not for a directory, we return error.
765  1       */

767  1      if (ntnp == NULL || !(ntnp->tn_flags & TNF_ISDIR))
768  2      {
769  2          return(EP_RB_RECOVER_INVALDIR);
770  1      }
```

```
772  1      /*
773  1       * Even though no error was returned, we have to apply the final
774  1       * permission check ourselves.
775  1       */

777  1      if (0 == err)
778  2      {
779  2           if (!rcx_permchk(rcptr, PERMCHK_R|PERMCHK_X, ntnp, (
                                                    rbcat_elem_t **)NULL))
780  3           {
781  3               return(EP_RB_RECOVER_PERMISSION_DENIED);
782  2           }

784  2           if (NULL != dirName)
785  3           {
786  3               rcptr->rc_pwd = esl_strdup(dirName);
787  3               if (NULL == rcptr->rc_pwd)
788  4               {
789  4                   rec_api_log_csm(SUB_CSM_NOMEM, NULL);
790  4                   return(EP_RB_RECOVER_NOMEM);
791  3               }
792  2           }
793  2           else
794  3           {
795  3               return EP_RB_RECOVER_INVALDIR;
796  2           }

798  2          mcat_set_current_dir(rcptr->rc_mcp, ntnp);
799  2           if (NULL != save_current_pwd)
800  3           {
801  3              free(save_current_pwd);
802  2           }
803  2          return(E_SUCCESS);
804  1      }

806  1      return(EP_RB_RECOVER_MCAT_ERR);
807    }        /* ChDir */
```



```
810    /*
811     * Function: ConcatFullName()
812     *
813     * Function Description:
814     *       Given the ptr to a tree_node which is for a directory that does
815     *       not have a valid corresponding catalog entry, this function
816     *       concatenates its full pathname by traversing the tree_node chain
817     *       upwards all the way to the root dir. It returns the ptr to
818     *       the concatenated full pathname string in the provided string
819     *       ptr.
820     *
821     *       Note that this function malloc the buffer for the full pathname
822     *       string. It is the caller's responsibility to free the memory
823     *       via the free() call.
824     *
825     * Parameters:
826     *      rcptr    - (I) Pointer to the restore context
827     *       tnPtr   - (I) ptr to the tree_node
828     *       fname   - (O) ptr to a pathname string ptr
829     *
830     * Return Codes:
831     *       E_SUCCESS                - operation is successful;
832     *       EP_RB_RECOVER_BAD_TREENODE - invalid treenode;
833     *       EP_RB_RECOVER_NOMEM - malloc failed;
834     */

836    static eerrno_ty
837    ConcatFullName( restore_context *rcptr,
838                    tree_node *tnPtr,
839                    char **fname )
840  1 {
841  1      tree_node *ptnPtr;   /* parent tree_node ptr */
842  1      char *fnPtr = NULL;  /* ptr to string buffer of full pathname */
843  1      char *sPtr = NULL;
844  1      size_t plen;
845  1      size_t clen;


848  1      /*
849  1       * Some basic validations.
850  1       */

852  1      if (NULL == tnPtr)
853  2      {
854  2           return(EP_RB_RECOVER_BAD_TREENODE);
855  1      }

857  1      if (!(tnPtr->tn_flags & TNF_ISDIR))
858  2      {
859  2           return(EP_RB_RECOVER_BAD_TREENODE);
860  1      }

862  1      if (NULL == tnPtr->tn_name)
863  2      {
864  2           return(EP_RB_RECOVER_BAD_TREENODE);
865  1      }

867  1      /*
868  1       * If the tnPtr points to the tree_node for the root dir,
869  1       * then simply return "/" as the full pathname.
870  1       */
```

```
872 1     if (tnPtr->tn_flags & TNF_ROOTDIR)
873 2     {
874 2          fnPtr = malloc(2);
875 2          if (NULL == fnPtr)
876 3          {
877 3              rec_api_log_csm(SUB_CSM_NOMEM, NULL);
878 3              return(EP_RB_RECOVER_NOMEM);
879 2          }
880 2          fnPtr[0] = '/';
881 2          fnPtr[1] = 0;
882 2          *fname = fnPtr;
883 2          return(E_SUCCESS);
884 1     }

886 1     clen = strlen(tnPtr->tn_name);

888 1     /*
889 1      * Alloc buffer for the node's name string (including the NULL
890 1      * terminator) plus a leading '/'. Then copy the current node's
891 1      * name into it with a leading '/'.
892 1      */

894 1     fnPtr = malloc(clen + 2);
895 1     if (NULL == fnPtr)
896 2     {
897 2         rec_api_log_csm(SUB_CSM_NOMEM, NULL);
898 2          return(EP_RB_RECOVER_NOMEM);
899 1     }

901 1     fnPtr[0] = '/';
902 1     memcpy(fnPtr+1, tnPtr->tn_name, clen);
903 1     fnPtr[1+clen] =  0;

905 1     ptnPtr = tnPtr->tn_parent;

907 1     /*
908 1      * Traverse the tree upwards until we hit the root dir
909 1      */

911 1     while (!(ptnPtr->tn_flags & TNF_ROOTDIR))
912 2     {
913 2          if (NULL == ptnPtr->tn_name)
914 3          {
915 3              return(EP_RB_RECOVER_BAD_TREENODE);
916 2          }
917 2          plen = strlen(ptnPtr->tn_name);
918 2          clen = strlen(fnPtr);

920 2          /*
921 2           * Alloc buffer for concatenating the parent node name
922 2           * and the child node name, plus a leading '/' and a
923 2           * NULL-terminator
924 2           */

926 2          sPtr = fnPtr;   /* we'll need to free this buffer later */
927 2          fnPtr = malloc(plen + clen + 2);
928 2          if (NULL == fnPtr)
929 3          {
930 3              rec_api_log_csm(SUB_CSM_NOMEM, NULL);
931 3              free(sPtr);
932 3              return(EP_RB_RECOVER_NOMEM);
933 2          }

935 2          fnPtr[0] = '/';
936 2          memcpy(fnPtr+1, ptnPtr->tn_name, plen);
```



```
937 2          memcpy(fnPtr+(1+plen), sPtr, clen);
938 2          fnPtr[1+plen+clen] = 0;

940 2          free(sPtr);

942 2          ptnPtr = ptnPtr->tn_parent;
943 1     }

945 1     *fname = fnPtr;

947 1     return(E_SUCCESS);
948  }         /* ConcatFullName */
```

```
950  /*
951   * Function: FindRealNode()
952   *
953   * Function Description:
954   *       Given the ptr to a tree_node which is for a directory that does
955   *       not have a valid corresponding catalog entry, this function
956   *       traverse down the directory until it finds the subdir that has
957   *       a valid catalog entry or until it reaches the end and fails. If
958   *       it succeeds, the full pathname of the valid directory is
959   *       returned.
                  The caller should use esl_strdup to copy the pathname.
960   *
961   * Parameters:
962   *       tnPtr    - (I) ptr to the tree_node
963   *       fname    - (O) ptr to a pathname string ptr
964   *
965   * Return Codes:
966   *       E_SUCCESS                - operation is successful;
967   *       EP_RB_RECOVER_BAD_TREENODE - invalid treenode;
968   */

970  static eerrno_ty
971  FindRealNode(tree_node *tnPtr,
972               char **fname)
973  1 {
974  1     tree_node *parentPtr;
975  1     tree_node **dir_ele;
976  1     tree_node *tnp;
977  1     int count;
978  1     int errnum;
979  1     rbtree_elem_t tree_elem;
980  1     rbcat_elem_t catElem;


983  1     if (NULL == tnPtr)
984  2     {
985  2          return(EP_RB_RECOVER_BAD_TREENODE);
986  1     }

988  1     /*
989  1      * The tree_node we are dealing with must be the first level
990  1      * down from the root dir "/". If it's not, then return error.
991  1      * If the tree_node itself is not a directory, then it's also
992  1      * an error.
993  1      */

995  1     parentPtr = tnPtr->tn_parent;
996  1     if (!(parentPtr->tn_flags & TNF_ROOTDIR))
997  2     {
998  2          return(EP_RB_RECOVER_BAD_TREENODE);
999  1     }

1001 1     if (!(tnPtr->tn_flags & TNF_ISDIR))
1002 2     {
1003 2          return(EP_RB_RECOVER_BAD_TREENODE);
1004 1     }

1006 1     tnp = tnPtr;

1008 1     /*
1009 1      * Traverse down the dir until a node is found that has a valid
1010 1      * catalog element.
```



```
1011 1      */

1013 1     for (;;)
1014 2     {
1015 2          /*
1016 2           * Looking for the top dir that has a valid catalog
1017 2           * entry.
1018 2           */

1020 2          count = mcat_array_children(rcp->rc_mcp, tnp, &dir_ele);
1021 2          tnp = *dir_ele;
1022 2          mcat_array_rls(rcp->rc_mcp, dir_ele);

1024 2          if (NULL == tnp)
1025 3          {
1026 3              return(EP_RB_RECOVER_BAD_TREENODE);
1027 2          }

1029 2          if (!(tnp->tn_flags & TNF_ISDIR))
1030 3          {
1031 3              return(EP_RB_RECOVER_BAD_TREENODE);
1032 2          }

1034 2          errnum = tlmno2recs(rcp,
1035 2                              tnp->tn_index,
1036 2                              tnp->tn_mcplane,
1037 2                              &tree_elem,
1038 2                              &catElem,
1039 2                              fname);

1041 2          if ((errnum == 0) && (fname != NULL))
1042 3          {
1043 3              return(E_SUCCESS);
1044 2          }

1046 2          if (count != 1)
1047 3          {
1048 3              return(EP_RB_RECOVER_BAD_TREENODE);
1049 2          }
1050 1     }
1051  }         /* FindRealNode */
```

```
1053    int
1054    fill_client_dirtop2(struct rbc_configs *rconfig,
1055                        boolean_ty InPlace,
1056                        char *directory,
1057                        char *workitem_name,
1058                        char *dest_hostname,
1059                        char **client_dirtop)

1061  1 {
1062  1     char buf[4096];
1063  1     RBC_WORKGROUP *pwg;
1064  1     RBC_WORKITEM *pwi = NULL;
1065  1     boolean_ty cross_recover;

1067  1     if ((NULL == workitem_name) ||
1068  1       (NULL == rconfig) ||
1069  1       (NULL == client_dirtop))
1070  2     {
1071  2       return -1;
1072  1     }

1074  1     for (pwg = rconfig->pgrouplist; NULL != pwg, NULL == pwi;
                                                          pwg = pwg->next)
1075  2     {
1076  2         for (pwi = pwg->pwilist; NULL != pwi; pwi = pwi->next)
1077  3         {
1078  3             if (0 == strcmp(pwi->name, workitem_name))
1079  4             {
1080  4                 break;
1081  3             }
1082  2         }
1083  1     }

1085  1     if (NULL == pwi)
1086  1          return -1;       /* didn't find workitem */


1089  1     /*
1090  1      * if the dirtop already specifies a netware client
1091  1      * target lets return an error.
1092  1      */

1094  1     if (*client_dirtop != NULL
1095  1         && NULL != strchr(*client_dirtop, ':'))
1096  2     {
1097  2         return -1;
1098  1     }

1100  1     /*
1101  1      * To test if this is a cross recovery compare the work items
1102  1      * configuration record field sysname (hostname) with the incoming
1103  1      * dest_hostname
1104  1      */

1106  1     cross_recover = ((NULL != pwi->sysname) && (
                                                 NULL != dest_hostname)) &&
1107  1                     (0 != strcmp(pwi->sysname, dest_hostname));

1109  2     {
1110  2       char *temp_char="";
1111  2       if(NULL != pwi && NULL != pwi->nw_clnt_target)
1112  3       {
1113  3          if(cross_recover)
1114  3            temp_char = dest_hostname;
1115  3          else
```



```
1116  3            temp_char = pwi->nw_clnt_target;
1117  2       }
1118  2       sprintf(buf, "%s%s\"%s\"",
1119  2                temp_char,
1120  2                ((
                      NULL != pwi && NULL != pwi->nw_clnt_target) ? ":" : ""),
1121  2                ((NULL != directory) ? directory : "/"));

1123  1     }

1125  1     if (*client_dirtop != NULL)
1126  2     {
1127  2         free (*client_dirtop);
1128  1     }

1130  1     *client_dirtop = esl_strdup(buf);
1131  1     if (NULL == *client_dirtop)
1132  2     {
1133  2       return -1;
1134  1     }
1135  1     return 0;

1137    }       /* fill_client_dirtop2 */
```

```
1142  boolean_ty GetClientTypeFromConfig(struct rbc_configs *config,
1143                                     char *workitem_name,
1144                                     char *wi_type)
1145 1 {
1146 1   boolean_ty wi_found = FALSE;
1147 1   RBC_WORKGROUP *wgp;
1148 1   RBC_WORKITEM *wip;

1150 1   if(NULL == workitem_name)
1151 2   {
1152 2     return FALSE;
1153 1   }

1155 1   for (wgp = config->pgrouplist;
1156 1        (wgp != NULL) && (FALSE == wi_found) ;
1157 1        wgp = wgp->next)
1158 2   {
1159 2     RBC_WORKITEM *wip;

1161 2     for (wip = wgp->pwilist; wip != NULL; wip = wip->next)
1162 3     {
1163 3       if((NULL != wip->name) &&
1164 3           (strcmp(wip->name, workitem_name) == 0))
1165 4       {
1166 4          *wi_type = wip->wi_type;
1167 4          wi_found = TRUE;
1168 4          break;
1169 3       }
1170 2     }
1171 1   }
1172 1   return wi_found;

1174  }
```



```
1177  /*****************************************************************
1178   **
1179   **     RSTLL_AddVolumeToList
1180   **
1181   **     Adds an entry to the ebvl_volidlist_ty list in the restore
                                                                  context,
1182   **     from its volid.
1183   **
1184   **  INPUTS:
1185   **
1186   **     rcptr    - Pointer to the restore context
1187   **     volid    - pointer to char string (
                                      ASCII hex) representation of volid
1188   **
1189   **  OUTPUTS:
1190   **     none, but ebvllist in restore context will be updated.
1191   **
1192   **  RETURN VALUE:
1193   **
1194   **     E_SUCCESS,
                EP_RB_RECOVER_BAD_ARGS or EP_RB_RECOVER_NO_VOLUME_DATA
1195   **
1196   **  IMPLICIT INPUTS:
1197   **
1198   **     volid must be a valid volume ID
1199   **
1200   *****************************************************************/

1202  eerrno_ty RSTLL_AddVolumeToList(
                              IN restore_context *rcptr, IN char *volid )
1203 1 {
1204 1         eperrno                 err;
1205 1         eerrno_ty       ret = E_SUCCESS;
1206 1         ebvl_volidlist_ty       *new_entry;
1207 1         ebvl_volidlist_ty       *last_entry;
1208 1         volumeID_ty     internalVol;

1210 1         if (NULL == volid || NULL == rcptr)
1211 1                 return EP_RB_RECOVER_BAD_ARGS;

1213 1         /* get volume id in internal format */
1214 1         rvmvolm_ascii_to_volumeID( &internalVol, volid );

1216 1         /* find end of list for ebvl_add_volid_to_list */
1217 1         if (NULL != (last_entry = rcptr->ebvllist))
1218 1                 for (; NULL != last_entry->next;
                                        last_entry = last_entry->next)
1219 1                         ;       /* null statement */

1221 1         err = ebvl_add_volid_to_list( &rcptr->ebvllist,
1222 1                                       &last_entry,
1223 1                                       internalVol );
1224 1         if ( (E_SUCCESS != err)
1225 1           || (NULL == (new_entry =
1226 1                       ebvl_is_volid_in_list(
                                    &internalVol, rcptr->ebvllist ))))
1227 2         {
1228 2             rbe_internal_error_sub(
                            err, "Error populating volume list entry" );
1229 2             ret = EP_RB_RECOVER_NO_VOLUME_DATA;
1230 1         }
1231 1         else
1232 2         {
1233 2             INCREMENT_VOLID_BFILE_COUNT (new_entry);
```

```
                                                        /* make sure used */
1234  1         }

1236  1         return ret;
1237    }
```



```
1240    /*****************************************************************
1241     **
1242     **     RSTLL_RemoveVolumeFromList
1243     **
1244     **     Removes an entry from the ebvl_volidlist_ty list in the
                                                                  restore context,
1245     **     using a volid to find it.
1246     **
1247     **  INPUTS:
1248     **
1249     **     rcptr     - Pointer to the restore context
1250     **     volid    - pointer to char string (
                                          ASCII hex) representation of volid
1251     **
1252     **  OUTPUTS:
1253     **     none, but ebvllist in restore context will be updated.
1254     **
1255     **  RETURN VALUE:
1256     **
1257     **     E_SUCCESS,
                    EP_RB_RECOVER_BAD_ARGS or EP_RB_RECOVER_NO_VOLUME_DATA
1258     **
1259     **  IMPLICIT INPUTS:
1260     **
1261     **     volid must be a valid volume ID
1262     **
1263     *****************************************************************/

1265    eerrno_ty RSTLL_RemoveVolumeFromList( IN restore_context *rcptr,
1266                                          IN char *volid )
1267  1 {
1268  1         volumeID_ty     internalVol;
1269  1         eerrno_ty       ret = E_SUCCESS;
1270  1         ebvl_volidlist_ty *vol_entry;

1272  1         if (NULL == volid || NULL == rcptr)
1273  1                 return EP_RB_RECOVER_BAD_ARGS;

1275  1         /* get volume id in internal format */
1276  1         rvmvolm_ascii_to_volumeID( &internalVol, volid );

1278  1         if (NULL == (vol_entry =
1279  1                     ebvl_is_volid_in_list(
                                        &internalVol, rcptr->ebvllist )))
1280  2         {
1281  2             ret = EP_RB_RECOVER_NO_VOLUME_DATA;
1282  2             rbe_internal_error_sub(
                                ret, "Error removing volume list entry" );
1283  1         }

1285  1         else
1286  2         {
1287  2                 /* decrement use count till zero */
1288  2                 while (DOES_VOLID_HAVE_MKD_FILES( vol_entry ))
1289  2                         DECREMENT_VOLID_BFILE_COUNT( vol_entry );
1290  1         }

1292  1         return ret;
1293    }
```

```
1295  /*****************************************************************
1296   **
1297   **      RSTLL_FreeVolidList
1298   **
1299   **      Frees a list of ebvl_volidlist_ty's
1300   **
1301   **  INPUTS:
1302   **      vol_list_ptr - address of the starting ebvl_volidlist_ty entry
1303   **
1304   **  OUTPUTS:
1305   **      none
1306   **
1307   **  RETURN VALUE:
1308   **
1309   **      none
1310   **
1311   **  IMPLICIT INPUTS:
1312   **
1313   **
1314   *****************************************************************/

1316  void RSTLL_FreeVolidList( IN ebvl_volidlist_ty *vol_list_ptr )
1317 1{
1318 1          ebvl_volidlist_destructor( vol_list_ptr, EBVL_DESTROY_ALL );
1319 1          return;
1320  }
```

```
/*****************************************************************************
 **
 ** File Name:   RSLmarkunm.c
 **
 ** Copyright (c) 1998,1999 by EMC Corporation.
 **
 ** Purpose:
 **       This module contains the Restore Service Library functions to mark and
 **       unmark objects for restoral.
 **
 ** Table of Contents:
 ** ------------------
 **       Service Library Functions:
 **               RSTSL_MarkObject
 **               RSTSL_UnmarkObject
 **
 **       Internal Functions:
 **        int    allowed_to_mark(
 **                       struct recover_context *rcx, tree_node *tnp,
 **                              char *name_if_known,
 **                                      rbcat_elem_t *clmp)
 ** static int    mtx(struct mcat_applyfunc_args *ctxp)
 ** static int    umtx(struct mcat_applyfunc_args *ctxp)
 ** static int    mark_tree(
 **                       struct recover_context *rcx, tree_node *tnp,
 **                        char *name_if_known)
 ** static int    unmark_tree(
 **                       struct recover_context *rcx, tree_node *tnp)
 **
 **
 ** Compile-Time Options:
 **               This section must list any compile time definitions
 **               which will affect this header.
 **
 **
 ** NOTE: Part of this module is adapted from:
 ** server/libs/recover/grandfathered/cmd_markunmark.c
 ** It contains mainly support routines needed by the mark and unmark
 ** API functions.
 **
 *****************************************************************************
                                                                        *****/



/* The following provides an RCS id in the binary that can be located
 ** with the what(1) utility.  The intent is to keep this short.
 */

#ifndef lint
static char RCS_id [] = "$RCSfile$ "
                        "$Revision$ "
                        "$Date$" ;
#endif



/*
 * Feature test switches.
 * Standard defines required to turn on OS features go here.
 *
 * The following is required for code that uses POSIX API's.
 * Remove for non-POSIX, non-portable code.
 */
```



```
#define  _POSIX_SOURCE 1



/*
 * System headers.
 */


/*
 * Epoch headers.
 */
#include <eb/eb_port.h>
#include <eb/rb_log.h>


/*
 * Local headers
 */
#include <RSLinterns.h>
#include <RSLrbrmain.h>



/*
 * #defines, structures, typedefs local to this source file
 */

struct mark_context
{
    boolean_ty no_badfiles;      /* don't mark bad files */
    boolean_ty no_descend;
                                 /* don't descend dirs, just mark them */
    ulong_t n_this;
    ulong_t n_before;
};

static int
mark_tree(struct recover_context *rcx,
          tree_node *tnp,
          char *name_if_known);

static int
unmark_tree(struct recover_context *rcx,
            tree_node *tnp);



/*
 * External declarations
 */

/*
 * Global variables for progress of marks/unmarks
 */
static RSTSL_MarkProgressProc global_progress_cb;
static boolean_ty global_was_cancelled;
static unsigned long global_start_marks;

NEW_SRC_FILE();



/*
 * Local function prototypes
 */

static int    mtx(struct mcat_applyfunc_args *ctxp);
static int    umtx(struct mcat_applyfunc_args *ctxp);
static void MarkUnmarkDebugLogEcho(tree_node *tnp);
```

```
125 static int check_parent_perms(struct recover_context *rcx,
126                                char *name_if_known, char *errorbuff);
```



```
128  /*********************************************
129   *
130   * allowed_to_mark:
131   *
132   *  Does the recovering user have permission to a file
133   *  for recovery.
134   *
135   * returns
136   *  1 if allowed to mark, 0 if permission is denied.
137   *
138   * Parameters:
139   *  rcx (I)           -- recover_context
140   *  tnp (I)           -- tree node pointer
141   *  name_if_known (I) -- name of file
142   *  clmp (I)          -- catalog elem
143   *********************************************/

145  int
146  allowed_to_mark(struct recover_context  *rcx,
147                  tree_node               *tnp,
148                  char                    *name_if_known,
149                  rbcat_elem_t            *clmp)
150 1 {
151 1     int ok = 1;

153 1     /*
154 1      * Modification for bug no OSGsw 23406
155 1      * perm_reason initialised to empty string
156 1      */
157 1     char perm_reason[256] = "";
158 1     /*
159 1      * Modifications for bug no OSGsw23406 end
160 1      */


163 1     if (S_ISREG(clmp->ce_mode))
164 2     {
165 2          /*
166 2           *  For regular files read permision required.
167 2           */

169 2          ok = rcx_permchk(rcx, PERMCHK_R, tnp, &clmp);
170 1     }
171 1     else if (S_ISDIR(clmp->ce_mode))
172 2     {
173 2          /*
174 2           *  For directories execute and read permision required.
175 2           */

177 2          ok = rcx_permchk(rcx, PERMCHK_R|PERMCHK_X, tnp, &clmp);
178 1     }
179 1     else if (S_ISBLK(clmp->ce_mode) || S_ISCHR(clmp->ce_mode))
180 2     {
181 2          /*
182 2           *  For char and block devices,
                                        only super-user can extract these.
183 2           */

185 2          ok = (rcx->rc_effective_uid == 0);
186 2          strcpy(perm_reason, "to recover you must be super-user");
187 1     }
188 1     else
189 2     {
190 2          /*
191 2           *  For unexpected file types no permission checks and we log
                                                              this one.
```

```
192  2            */

194  2          char pathbuf[EB_MAXPATHLEN];
195  2          char *pbp = pathbuf;


198  2          tn_getpath(tnp, &pbp);

200  2          ok = 1;
201  2          rbe_log_stats(0,
202  2                        "Unexpected File Type,
                                   no permission checks to mark file %s", pbp);
203  1       }

205  1       if (!ok)
206  2       {

208  2       /*
209  2        * Modifications for bug no OSGsw23406
210  2        */
211  2            if (NULL != name_if_known)
212  3            {
213  3                rbe_log_stats(0,
214  3                              "Permission denied for file \"%s\" %s.",
215  3                              name_if_known, perm_reason);
216  2           }
217  2            else
218  3            {
219  3                char pathbuf[EB_MAXPATHLEN];
220  3                char *pbp = pathbuf;


223  3                tn_getpath(tnp, &pbp);
224  3                rbe_log_stats(0,
225  3                              "Permission denied for file \"%s\" %s.",
226  3                              pbp, perm_reason);
227  2            }
228  2       /*
229  2        * Modifications for bug no OSGsw23406 end
230  2        */

232  1       } /* end of if not ok */

234  1       return ok;
235     }         /* end of allowed_to_mark() */
```



```

237     /*********************************************
238      *
239      * mtx:
240      *
241      *
242      *********************************************/

244     static int
245     mtx(struct mcat_applyfunc_args *ctxp)
246  1  {
247  1      struct recover_context *rcx =
248  1           LINTABLE_CAST(struct recover_context *, ctxp->arg);

250  1      return mark_tree(rcx, ctxp->tnp, (char *)NULL);
251     }         /* end of mtx() */
```

```
253  /*********************************************
254   *
255   * umtx:
256   *
257   *
258   *********************************************/

260  static int
261  umtx(struct mcat_applyfunc_args *ctxp)
262 1 {
263 1     struct recover_context *rcx =
264 1          LINTABLE_CAST(struct recover_context *, ctxp->arg);

266 1     return unmark_tree(rcx, ctxp->tnp);
267  }         /* end of umtx() */
```



```
269  /*
270   * The Below are static variables global to this module api_markunmark.c
271   *  They get initialized in RSTSL_MarkObject to 0.
272   *  They get incremented in mark_tree() when the conditions of mark are
273   *  encountered.
274   *  They get assigned to the output parameters to RSTSL_MarkObject after
275   *  the mark_tree call.
276   */

278  static unsigned long PermissionDeniedFileCount;
279  static unsigned long BADDATAFileCount;

281  /*********************************************
282   *
283   * mark_tree:
284   *
285   *  Provides mark function that is based on the current mcat
286   *  recover context. The mark operation has hidden parameters
287   *  in the mark_context struct.
288   *
289   * Globals:
290   *  PermissionDeniedFileCount and BADDATAFileCount
291   *  are statics global to this module.
292   *
293   * returns 0 for success always.
294   *
295   * Parameters:
296   *  rcx (I) -- recover_context
297   *  tnp (I) -- tree node pointer
298   *  name_if_known (I) -- name of imput file.
299   *********************************************/

301  static int
302  mark_tree(struct recover_context *rcx,
303            tree_node *tnp,
304            char *name_if_known)
305 1 {
306 1     struct mark_context *mcp =
307 1          LINTABLE_CAST(struct mark_context *, rcx->rc_cmd_context);

309 1     eperrno ep_status = E_SUCCESS;


312 1   /*
313 1    * Check to see if the mark operation has been cancelled,
314 1    * If so, return without marking.
315 1    *
316 1    * This is due to the recursiveness of the mcat_apply2children
317 1    * function to stop any further processing.
318 1    */

320 1   if (global_was_cancelled)
321 2   {
322 2     return (0);
323 1   }

325 1     /*
326 1      * Only do work if element not already marked.
327 1      */

329 1     if (! TMARKED(rcx->rc_marks, tnp))
330 2     {
```

```
331 2           rbcat_elem_t clm;

333 2           /*
334 2            * get catalog record
335 2            */

337 2           mcat_getcatlm(rcx->rc_mcp, tnp, &clm);

339 2           /*
340 2            * Skip file if it's bad and user wants to ignore bad files
341 2            */

343 2           if (mcp->no_badfiles && (CESTAT_BADDATA & clm.ce_status))
344 3           {
345 3               char pathbuf[EB_MAXPATHLEN];
346 3               char *pbp = pathbuf;


349 3               tn_getpath(tnp, &pbp);
350 3               rbe_log_stats(
                       0, "The file %s has BADDATA and was not marked", pbp);
351 3               BADDATAFileCount++;
352 3               return 0;
353 2           }

355 2           if (CESTAT_BADDATA & clm.ce_status)
356 3           {
357 3               char pathbuf[EB_MAXPATHLEN];
358 3               char *pbp = pathbuf;


361 3               tn_getpath(tnp, &pbp);
362 3               rbe_log_stats(
                             0, "The file %s is BADDATA and was marked", pbp);
363 3               BADDATAFileCount++;
364 2           }

366 2           /*
367 2            * Check permissions.  Try to avoid calling permission
368 2            * functions by making some trivial checks in line
369 2            * (owner can always mark; root can always mark).
370 2            */

372 2           if (rcx->rc_effective_uid != (uid_t)clm.ce_owner
373 2               && rcx->rc_effective_uid != 0
374 2               && ! allowed_to_mark(rcx, tnp, name_if_known, &clm))
375 3           {
376 3               /*
377 3                * logging done in allow_to_mark
378 3                */

380 3               PermissionDeniedFileCount++;
381 3               return 0;
382 2           }

384 2           /*
385 2            * 1) Setting mark this bitfile for recovery bit.
386 2            * 2) incrementing marks for the plane
387 2            * 3) incrementing total marks
388 2            * 4) incrementing the operated on bitfile count for call to
                                                                   mart_tree
389 2            */

391 2           TSETMARK(rcx->rc_marks, tnp);
392 2           rcx->rc_nmarks_by_plane[-tnp->tn_mcplane]++;
393 2           rcx->rc_nmarks_total++;
```



```
394 2           mcp->n_this++;

396 2           /*
397 2            * To avoid double logging on mark, do not log BADDATA files.
398 2            */

400 2           if (! CESTAT_BADDATA & clm.ce_status)
401 3           {
402 3                MarkUnmarkDebugLogEcho(tnp);
403 2           }

405 2           /*
406 2            * If this is a DS_NONE record and has no catlm,
407 2            * then we need to remember where we can find the
408 2            * actual catlm information for this node.
409 2            */

411 2           if (tnp->tn_tlm_catlm == -1)
412 3           {
413 3               eerrno_ty ret_status;


416 3               if (E_SUCCESS != (ret_status = add_dsnone(rcx, tnp)))
417 4               {
418 4                   return ret_status;
419 3               }
420 2           }

422 2           /*
423 2            * If were marking lets add the bfile to the volid list.
424 2            * if needed adding the volume to the list and incrementing
425 2            * the bfile dependacy count.
426 2            */

428 2           rcx->ebvllist = ebvl_add_bfile_to_volid_count (
                                                              rcx-> ebvllist,
429 2                                                  (ebfs_uid_ty*) &(
                                                          clm.ce_bitfileID),
430 2                                                  &ep_status);
431 2           if (E_SUCCESS != ep_status)
432 3           {
433 3               /*
434 3                * This error does not effect the recover,
                                                              but lets log it
435 3                */

437 3               rbe_internal_error( ep_status,
438 3                   "NOTE: in ebvl_add_bfile_to_volid_count(
                                                             ) for bitfile %s",
439 3                   eb_ebfsid2ascii(&clm.ce_bitfileID, NULL,
440 3                                   EB_EBFSID2ASCII_WITH_DOTS) );
441 2           }

443 2           /*
444 2            * if the summary is valid, update it for this bfile.
445 2            */

447 2           if (rcx->rc_mark_summary_valid)
448 3           {
449 3               add_to_summary(&rcx->rc_mark_summary, &clm);
450 2           }

452 2           /*
453 2            * If we have reached the boundary for reporting progress
454 2            * prior to marking this file, call the progress routine
455 2            * and verify it is OK to continue
```

```
456 2            */
457 2           if (((
                    rcx->rc_nmarks_total - global_start_marks) & 0x3ff) == 0)
458 3           {
459 3             /* Don't bother on 0 */
460 3             if ((rcx->rc_nmarks_total - global_start_marks) > 0)
461 4             {
462 4               /* If callback returns non-zero,
                                        flag this operation as cancelled */
463 4               if (global_progress_cb (
                              rcx->rc_nmarks_total - global_start_marks))
464 5               {
465 5                 /* Set the cancelled flag */
466 5                 global_was_cancelled = TRUE;

468 5                 /* Get out, since user cancelled */
469 5                 return (0);
470 4               }
471 3             }
472 2           }

474 1       } /* end of if (!TMARKED()) */

476 1       if (tnp->tn_flags & TNF_KNOWN_NOCHILDREN)
477 2       {
478 2            return 0;
479 1       }

481 1      /*
482 1       * Should we descend to chidren.
483 1       */

485 1       if (!mcp->no_descend)
486 2       {
487 2            (void)mcat_apply2children(rcx->rc_mcp, tnp, mtx, (char *)rcx);
488 1       }
489 1       return 0;
490     }         /* end of mark_tree() */
```



```
493     /*
494      * The Below are static variables global to this module api_markunmark.
                                                                          c
495      *  They get initialized in EDMRST_UnmarkObject to 0.
496      *  They get incremented in unmark_tree(
                                                ) when the badfiles are encountered.
497      *  They get assigned to the output parameters to EDMRST_UnmarkObject
                                                                        after
498      *  the unmark_tree call.
499      */

501     static long BADDATAFileUnmarkCount;

503     /*********************************************
504      *
505      * unmark_tree:
506      *
507      *  Provides unmark function that is based on the current mcat
508      *  recover context. The unmark operation has hidden parameters
509      *  in the mark_context struct.
510      *
511      * Globals:
512      *  BADDATAFileUnmarkCountis a static global to this module.
513      *
514      * returns 0 for success and -1 if there are no marks to unmark.
515      *
516      * Parameters:
517      *  rcx (I) -- recover_context
518      *  tnp (I) -- tree node pointer
519      *
520      *********************************************/

522     static int
523     unmark_tree(struct recover_context *rcx,
524                 tree_node *tnp)
525 1   {
526 1       struct mark_context *mcp =
527 1            LINTABLE_CAST(struct mark_context *, rcx->rc_cmd_context);

529 1       eperrno ep_status = E_SUCCESS;


532 1     /*
533 1      * Check to see if the unmark operation has been cancelled,
534 1      * If so, return without unmarking.
535 1      *
536 1      * This is due to the recursiveness of the mcat_apply2children
537 1      * function to stop any further processing.
538 1      */

540 1     if (global_was_cancelled)
541 2     {
542 2       return (0);
543 1     }

545 1       /*
546 1        * If there are no more things marked,
                                                  don't need to go any further
547 1        */

549 1       if (rcx->rc_nmarks_total == 0)
550 2       {
551 2            return -1;
552 1       }
```

```
555 1      /*
556 1       * unmark this guy
557 1       */

559 1      if (TMARKED(rcx->rc_marks, tnp))
560 2      {
561 2           rbcat_elem_t clm;

563 2           /*
564 2            * get catalog record
565 2            */

567 2           mcat_getcatlm(rcx->rc_mcp, tnp, &clm);

569 2           /*
570 2            * Skip file if we're only unmarking bad files
571 2            * and this one is NOT bad.
572 2            */

574 2           if (mcp->no_badfiles && !(CESTAT_BADDATA & clm.ce_status))
575 3           {
576 3               goto unmark_children;
577 2           }

579 2           /*
580 2            * Count bad files
581 2            */

583 2           if (CESTAT_BADDATA & clm.ce_status)
584 3           {
585 3               BADDATAFileUnmarkCount++;

587 3               /*
588 3                * Log if this is an unmark BADDATA files only unmark
                                                                        call
589 3                *
590 3                */

592 3               if (mcp->no_badfiles)
593 4               {
594 4                   char pathbuf[EB_MAXPATHLEN];
595 4                   char *pbp = pathbuf;


598 4                   tn_getpath(tnp, &pbp);
599 4                   rbe_log_stats(
                                0, "The BADDATA file %s was umarked", pbp);
600 3               }
601 2           }

603 2           /*
604 2            * 1) Clearing mark this bitfile for recovery bit.
605 2            * 2) decrementing marks for the plane
606 2            * 3) decrementing total marks
607 2            * 4) incrementing the unmarked bitfile count for call to
                                                                    unmark_tree
608 2            */

610 2           TCLRMARK(rcx->rc_marks, tnp);
611 2           rcx->rc_nmarks_by_plane[-tnp->tn_mcplane]--;
612 2           rcx->rc_nmarks_total--;
613 2           mcp->n_this++;

615 2           /*
```



```
616 2           * To avoid double logging on mark, do not log BADDATA files.
617 2           * if BadfileOnly has been set.
618 2           */

620 2           if (!(mcp->no_badfiles) && (CESTAT_BADDATA & clm.ce_status))
621 3           {
622 3               MarkUnmarkDebugLogEcho(tnp);
623 2           }

625 2           /*
626 2            * If were unmarking lets decrement the bfile to the volid
                                                                        list
627 2            * bfile dependacy count.
628 2            */

630 2           ep_status = ebvl_decr_bfile_to_volid_count (rcx-> ebvllist,
631 2                                                       (ebfs_uid_ty*) &(
                                                          clm.ce_bitfileID));
632 2           if (E_SUCCESS != ep_status)
633 3           {
634 3               /*
635 3                * This error does not effect the recover,
                                                              but lets log it
636 3                */

638 3               rbe_internal_error( ep_status,
639 3                   "NOTE: in ebvl_decr_bfile_to_volid_count(
                                                          ) for bitfile %s",
640 3                   eb_ebfsid2ascii(&clm.ce_bitfileID, NULL,
641 3                                   EB_EBFSID2ASCII_WITH_DOTS) );
642 2           }

644 2           /*
645 2            * if the summary is valid, update it for this bfile.
646 2            */

648 2           if (rcx->rc_mark_summary_valid)
649 3           {
650 3               sub_from_summary(&rcx->rc_mark_summary, &clm);
651 2           }

653 2           /*
654 2            * undo dsnone info saved, if any
655 2            */

657 2           if (tnp->tn_tlm_catlm == -1)
658 3           {
659 3               remove_dsnone(rcx, tnp);
660 2           }

662 2           /*
663 2            * If we have reached the boundary for reporting progress
664 2            * prior to unmarking this file, call the progress routine
665 2            * and verify it is OK to continue
666 2            */

668 2           if (((
                   global_start_marks - rcx->rc_nmarks_total) & 0x3ff) == 0)
669 3           {
670 3             /* Don't bother on 0 */
671 3             if ((global_start_marks - rcx->rc_nmarks_total) > 0)
672 4             {
673 4               /* If callback returns non-zero,
                                    flag this operation as cancelled */
674 4               if (global_progress_cb (
                                  global_start_marks - rcx->rc_nmarks_total))
```

```
675 5                {
676 5                  /* Set the cancelled flag */
677 5                  global_was_cancelled = TRUE;

679 5                  /* Get out, since user cancelled */
680 5                  return (0);
681 4                }
682 3              }
683 2            }
684 1      }

686 1  unmark_children:

688 1      if (tnp->tn_flags & TNF_KNOWN_NOCHILDREN)
689 2      {
690 2           return 0;
691 1      }

693 1      if (!mcp->no_descend)
694 2      {
695 2           (void)mcat_apply2children(rcx->rc_mcp, tnp, umtx, (
                                                                  char *)rcx);
696 1      }
697 1      return 0;
698    }   /* end of unmark_tree() */
```



```
700    static int
701    check_parent_perms(struct recover_context *rcx,
702                       char *name_if_known,
703                       char *errorbuff)
704 1  {
705 1      rbcat_elem_t clm;
706 1      rbcat_elem_t *clmp = &clm;
707 1      tree_node *tnp;
708 1      char      parent_path[PATH_MAX], *tmpptr;
709 1      int       ok = 1;


712 1      if (rcx->rc_effective_uid == 0)
713 2      {
714 2          return 0;
715 1      }

717 1      strcpy(parent_path, name_if_known);

719 1      if ((tmpptr = strrchr(parent_path, '/')) == NULL)
720 2      {
721 2           return 0;
722 1      }

724 1      do
725 2      {
726 2          if (tmpptr == parent_path)
727 3           {
728 3              return 0;
729 2           }

731 2          *tmpptr = 0;

733 2          tnp = mcat_lookup_path(rcx->rc_mcp, parent_path);

735 2          /*
736 2           * get catalog record
737 2           */

739 2          if (tnp != NULL)
740 3          {
741 3              mcat_getcatlm(rcx->rc_mcp, tnp, &clm);

743 3              if (rcx->rc_effective_uid != clm.ce_owner)
744 4               {
745 4                  ok = rcx_permchk(rcx, PERMCHK_R|PERMCHK_X, tnp, &clmp);
746 3               }
747 2          }
748 1      }
749 1      while (ok && (tmpptr = strrchr(parent_path, '/')) != NULL);

751 1      if (ok == 0 && errorbuff != NULL)
752 2      {
753 2          strcpy(errorbuff, parent_path);
754 1      }

756 1      return !ok;
757    }        /* check_parent_perms */
```

```
760  /*****************************************************************
761   * RSTSL_MarkObject()
762   *
763   *   The MarkObject operation takes a restorableObject and marks it, and
764   *   possibly its descendant files for restoral based on the input criteria.
765   *   The RSTSL_MarkObject call is an asynchronously executed operation
766   *   in the Restore Engine that performs the marking.  It stores its results
767   *   in buffers supplied by the caller, so that the synchronous RPC thread can
768   *   test for and retrieve results without another Restore Service Library
769   *   call.
770   *
771   * Parameters:
772   *
773   *  thisObject    (I) - The restoral object; can be a leaf object (e.g. a
774   *                      file), or a container object ( e.g., a directory).
775   *  time          (I) - ( optional) the backup time to perform the mark on --
776   *                      if not specified, uses currrently selected backup; if
777   *                      specified, leaves selected backup time unchanged
778   *  allowBadfiles (I) - allows marking of files of state BADDATA.
779   *  descend       ( I) - Should mark operation descend to operate on the content
780   *                      of container objects.
781   *  BadFilesCount (O) - returns the file count with BADDATA.
782   *  PermDenyFilesCount ( O) -- returns the file count with permission denied.
783   *  fileMarked    ( O) - return the total files marked after this mark occurred.
784   *  dirMarked     ( O) - return the total directories marked after this mark
785   *                      occurred.
786   *  otherMarked   ( O) - return the total "other" files marked after this mark.
787   *  progressCB    ( I) - pointer to callback function to report progress and
788   *                      test for cancellation
789   *
790   *****************************************************************/

792  eerrno_ty
793  RSTSL_MarkObject( struct RSTRPC_user_restorable_object *thisObject,
794                    time_t          backupTime,
795                    boolean_ty      allowBadFiles,
796                    boolean_ty      descend,
797                    unsigned long *BadFilesCount,
798                    unsigned long *PermDenyFilesCount,
799                    unsigned long *fileMarked,
800                    unsigned long *dirMarked,
801                    unsigned long *otherMarked,
802                    RSTSL_MarkProgressProc progressCB )
803  1 {
804  1     struct mark_context mc;
805  1     boolean_ty ssdb_opened = FALSE;
806  1     char   errorbuff[PATH_MAX];
807  1     int rtc;      /* return code from some internal routines */
```



```
808  1     time_t        save_time = 0;
809  1     u_hyper       lenMarkedFiles;
810  1     RSTRPC_backup_flags_ty        backup_flags = 0;

812  1     /*
813  1      *  Output parameters check pointers against NULL & intialized to zero
814  1      */

816  1     if (!fileMarked || !dirMarked || !otherMarked)
817  2     {
818  2          return EP_RB_RECOVER_BAD_ARGS;
819  1     }

821  1     *fileMarked = *dirMarked = *otherMarked=0;

823  1     if (!BadFilesCount || !PermDenyFilesCount)
824  2     {
825  2          return EP_RB_RECOVER_BAD_ARGS;
826  1     }

828  1     /*
829  1      * lets validate the restorableObject as an input parameter.
830  1      */

832  1      if (NULL == thisObject)
833  2      {
834  2          return EP_RB_RECOVER_BAD_ARGS;
835  1      }

837  1      if (thisObject->root.objName == NULL)
838  2      {
839  2           return (EP_RB_RECOVER_INVAL_OBJNAME);
840  1      }

842  1      switch (thisObject->root.objLevel)
843  2      {
844  2      case RSTRPC_container_type:
845  2      case RSTRPC_leaf_type:
846  2          /* make sure input object is from current TLO's backup app */
847  2           if (rcp->rc_backup_app != thisObject->root.backupApp)
848  2                return (EP_RB_RECOVER_INVALOP);
849  2           break;

851  2      case RSTRPC_tlo_type:
852  2           return (EP_RB_RECOVER_INVALOP);

854  2      default:
855  2           return (EP_RB_RECOVER_INVAL_OBJTYPE);
856  1      }

858  1     /* check if backup time must change */
859  1     if ( backupTime != (time_t)0 )
860  2     {
861  2          /* get current backup time; if different from requested time,
862  2           * then set requested time and save current time; at end, restore
863  2           * current time */

865  2          if (rcp->rc_backup_app != 0)
866  3          {
867  3                  rtc = rcp-> currentPIptr-> piFuncArray[ PIFuncIndexGetCurTime ]
868  3                                                        (rcp, &save_time);
869  3                  if (rtc == E_SUCCESS)
```

```
870 3                           rtc = rcp-> currentPIptr->
871 3                                   piFuncArray[
                                            PIFuncIndexSetBkupTime ]
872 3                                       (
                                    rcp, backupTime, backup_flags);
873 3          } else {
874 3                  rtc = RSTSL_GetCurrentBackupTime( &save_time );
875 3                  if (rtc == E_SUCCESS)
876 3                          rtc = RSTSL_SetBackupForTime (
                                        backupTime, backup_flags);
877 2          }
878 2          if (rtc != E_SUCCESS)
879 2                  return rtc;
880 1      }

882 1      /* if this isn't a network backup restore,
                                        call plugin to do mark */
883 1      if (rcp->rc_backup_app != 0)
884 2      {
885 2           rtc = rcp-> currentPIptr-> piFuncArray[ PIFuncIndexMark ]
886 2                       (
                          rcp, thisObject, allowBadFiles, descend,
887 2                          BadFilesCount, PermDenyFilesCount,
                                                    fileMarked,
888 2                          &lenMarkedFiles, dirMarked,
                                                    otherMarked,
889 2                          progressCB );

891 2           /* save mark summary in restore_context */
892 2           rcp->rc_mark_summary.nfiles = *fileMarked ;
893 2           rcp->rc_mark_summary.ndirs = *dirMarked;
894 2           rcp->rc_mark_summary.nother = *otherMarked;
895 2           rcp->rc_mark_summary.len_mkd_files.high = lenMarkedFiles.high;
896 2           rcp->rc_mark_summary.len_mkd_files.low = lenMarkedFiles.low;

898 2           if (save_time)
899 2                   rcp-> currentPIptr-> piFuncArray[
                                        PIFuncIndexSetBkupTime ]
900 2                                       (
                                    rcp, save_time, backup_flags);

902 2           return rtc;
903 1      }


906 1      /* Initial global marking/unmarking cancel flag to FALSE */
907 1      global_was_cancelled = FALSE;
908 1      global_progress_cb = progressCB;
909 1      global_start_marks = rcp->rc_nmarks_total;


912 1      mc.n_this = 0;
913 1      mc.n_before = rcp->rc_nmarks_total;

915 1      *BadFilesCount = *PermDenyFilesCount = 0;

917 1     /*
918 1      * The Below are static variables global to this module
                                                    api_markunmark.c
919 1      *  They get initialized in EDMRST_MarkObject to 0.
920 1      *  They get incremented in mark_tree(
                                        ) when the conditions of mark are
921 1      *  encountered.
922 1      *  They get assigned to the output parameters to EDMRST_MarkObject
                                                            after
923 1      *  the mark_tree call.
```



```
924 1      */

926 1      BADDATAFileCount = PermissionDeniedFileCount = 0;

928 1      rcp->rc_cmd_context = (char *)&mc;

930 1      /*
931 1       * input parameters
932 1       * 1) allowBadfiles --  does marking of Badfiles
933 1       * 2) descend --  does mark descend into the contents of
                                                        directories.
934 1       */

936 1      if (!allowBadFiles)
937 2      {
938 2           mc.no_badfiles = TRUE;
939 1      }
940 1      else
941 2      {
942 2           mc.no_badfiles = FALSE;
943 1      }

945 1      if (!descend)
946 2      {
947 2           mc.no_descend = TRUE;
948 1      }
949 1      else
950 2      {
951 2           mc.no_descend = FALSE;
952 1      }

954 1      /*
955 1       * Open up the saveset db during mark command.
956 1       */

958 1      rtc = ss_open_saveset_db();
959 1      if (0 != rtc)        /* ss_open_saveset_db failed */
960 2      {
961 2           return (EP_RB_RECOVER_CANT_OPEN_SSDB);
962 1      }

964 1      /*
965 1       * ssdb_opened was used to indicate whether or not
966 1       * ss_open_saveset_db was successful. It was set to TRUE
967 1       * when ss_open_saveset_db() succeeded, so we'll remember
968 1       * to call ss_close_saveset_db later. This is no longer
969 1       * necessary because we would have returned an error if
970 1       * ss_open_saveset_db failed. It's left in to minimize
971 1       * code changes.
972 1       */

974 1      ssdb_opened = TRUE;

976 1      /*
977 1       * validate restorable object with the current mcat context
978 1       */

980 2      {
981 2           /*
982 2            * Initialize tnp_thisObject to NULL so that if
                                                        mcat_lookup_path
983 2            * does not set it to NULL in the case of failure, the code
984 2            * can work correctly.
985 2            */
986 2           tree_node *tnp_thisObject = NULL;
```

```
988 2           tnp_thisObject = mcat_lookup_path(
                                    rcp->rc_mcp, thisObject->root.objName);
989 2           if (tnp_thisObject == NULL)
990 3           {
991 3                /*
992 3                 *   If we opened ssdb then lets close it!
993 3                 *
994 3                 *   restorable object must be corrupt mcat_lookup_path
                                                                    could not
995 3                 *   locate the tree node ptr for files
996 3                 */
997 3                if (ssdb_opened)
998 4                {
999 4                    ss_close_saveset_db();
1000 3               }
1001 3               return EP_RB_RECOVER_BAD_CONTEXT;
1002 2          }

1004 2          if (tnp_thisObject->tn_mcplane !=
1005 2              ((netBackupObjData *)(
                              thisObject->appData.data))->objtnMCplane)
1006 3          {
1007 3               /*
1008 3                * If we opened ssdb then lets close it!
1009 3                *
1010 3                * restorable object must be out of context, return error
1011 3                */

1013 3               if (ssdb_opened)
1014 4               {
1015 4                   ss_close_saveset_db();
1016 3               }
1017 3               return EP_RB_RECOVER_BAD_CONTEXT;
1018 2          }

1020 2          /*
1021 2           * If we made it here then thisObject is valid.
1022 2           */

1024 2          if (check_parent_perms(
                             rcp, thisObject->root.objName, errorbuff))
1025 3          {
1026 3               rbe_log_stats(0,
1027 3                             "Permission denied for file \"%s\" %s.",
1028 3                             thisObject->root.objName,
1029 3                             "parent directory permissions");
1030 2          }
1031 2          else
1032 3          {
1033 3               /*
1034 3                * mark_tree always returns 0
1035 3                */

1037 3               (void) mark_tree(
                           rcp, tnp_thisObject, thisObject->root.objName);
1038 2          }
1039 1     }

1041 1     /*
1042 1      * The Below are static variables global to this module
                                                          api_markunmark.c
1043 1      *  They get initialized in EDMRST_MarkObject to 0.
1044 1      *  They get incremented in mark_tree(
                                        ) when the conditions of mark are
1045 1      *  encountered.
1046 1      *  They get assigned to the output parameters to
```



```
                                                    EDMRST_MarkObject after
1047 1      *  the mark_tree call.
1048 1      */

1050 1     *BadFilesCount = BADDATAFileCount;
1051 1     *PermDenyFilesCount = PermissionDeniedFileCount;

1053 1     /*
1054 1      * if the summary is valid, set the total number of marked files,
1055 1      * directories, and "other" files. This is not intended to be the
1056 1      * number of files that resulted directly from the above mark_tree
                                                                    call.
1057 1      */

1059 1     if (rcp->rc_mark_summary_valid)
1060 2     {
1061 2          *fileMarked = rcp->rc_mark_summary.nfiles;
1062 2          *dirMarked = rcp->rc_mark_summary.ndirs;
1063 2          *otherMarked = rcp->rc_mark_summary.nother;
1064 1     }
1065 1     else
1066 2     {
1067 2          /*
1068 2           * This error is not fatal, the output variable *Marked,
1069 2           * will be zero, we should never get this error.
1070 2           */

1072 2          rbe_log_stats(
   0, "Internal error: mark summary not Valid in EDMRST_MarkObject()");
1073 1     }

1075 1     /*
1076 1      * If we opened it then lets close it!
1077 1      */

1079 1     if (ssdb_opened)
1080 2     {
1081 2          ss_close_saveset_db();
1082 1     }

1084 1     if (save_time)
1085 1          RSTSL_SetBackupForTime( save_time, backup_flags );

1087 1     return( E_SUCCESS );
1088 }         /* end of RSTSL_MarkObject () */
```

```
1090  /*****************************************************************
1091   * UnmarkObject
1092   *
1093   *   The UnmarkObject operation takes a restorableObject and unmarks
                                                                    it, and
1094   *   possibly its descendant files for restoral based on the input
                                                                  criteria.
1095   *   The RSTSL_UnmarkObject call is an asynchronously executed
                                                                  operation
1096   *   in the Restore Engine that performs the unmarking.
                                                            It returns unmark
1097   *   progress and tests for user cancelation through a callback
                                                                  function.
1098   *
1099   * UnmarkObject Parameters:
1100   *
1101   *  thisObject    (I) - The restoral object;
                                          can be a leaf object (e.g. a
1102   *                      file), or a container object (
                                                 e.g., a directory).
1103   *  backupTime    (I) - (
                      optional) the backup time to perform the unmark on --
1104   *                      if not specified,
                                    uses currrently selected backup; if
1105   *                      specified,
                                 leaves selected backup time unchanged
1106   *  BadFilesOnly  (
                     I) - allows unmarking ONLY of files of state BADDATA.
1107   *  descend       (
                    I) - Should unmark operation descend to operate on the
1108   *                      content of container objects.
1109   *  BadFilesCount (O) - returns the file count with BADDATA.
1110   *  fileMarked    (
           O) - return the total files marked after this mark occurred.
1111   *  dirMarked     (
                O) - return the total directories marked after this mark
1112   *                      occurred.
1113   *  otherMarked   (
              O) - return the total "other" files marked after this mark.
1114   *  progressCB    (
                I) - pointer to callback function to report progress and
1115   *                      test for cancellation
1116   *****************************************************************/

1118  eerrno_ty RSTSL_UnmarkObject(
                        struct RSTRPC_user_restorable_object *thisObject,
1119                              const time_t          backupTime,
1120                               const boolean_ty     BadFilesOnly,
1121                               const boolean_ty     descend,
1122                              ulong                 *BadFilesCount,
1123                              ulong                 *fileMarked,
1124                              ulong                 *dirMarked,
1125                              ulong                 *otherMarked,
1126                               RSTSL_MarkProgressProc progressCB )
1127 1 {
1128 1     struct mark_context mc;
1129 1     eerrno_ty     rtc;
1130 1     time_t        save_time = 0;
1131 1     u_hyper       lenMarkedFiles;
1132 1     RSTRPC_backup_flags_ty       backup_flags = 0;

1134 1     /* Initial global marking/unmarking cancel flag to FALSE */
1135 1     global_was_cancelled = FALSE;
1136 1     global_progress_cb = progressCB;
1137 1     global_start_marks = rcp->rc_nmarks_total;
```



```
1139 1     mc.n_this = 0;
1140 1     mc.n_before = rcp->rc_nmarks_total;

1142 1     rcp->rc_cmd_context = (char *)&mc;


1145 1    /*
1146 1     *  Output parameters get intialized to zero
1147 1     */

1149 1    /*
1150 1     *  Output parameters check pointers against NULL & intialized to
                                                                       zero
1151 1     */

1153 1     if (!fileMarked || !dirMarked || !otherMarked || !BadFilesCount)
1154 2     {
1155 2          return EP_RB_RECOVER_BAD_ARGS;
1156 1     }

1158 1     /*
1159 1      * lets validate the restorable object as an input parameter.
1160 1      */

1162 1     if (NULL == thisObject)
1163 2     {
1164 2         return EP_RB_RECOVER_BAD_ARGS;
1165 1     }

1167 1     if (thisObject->root.objName == NULL)
1168 2     {
1169 2          return(EP_RB_RECOVER_INVAL_OBJNAME);
1170 1     }

1172 1     switch (thisObject->root.objLevel)
1173 2     {
1174 2     case RSTRPC_container_type:
1175 2     case RSTRPC_leaf_type:
1176 2          /* make sure input object is from current TLO's backup app */
1177 2           if (rcp->rc_backup_app != thisObject->root.backupApp)
1178 2                 return (EP_RB_RECOVER_INVALOP);
1179 2          break;

1181 2     case RSTRPC_tlo_type:
1182 2          return (EP_RB_RECOVER_INVALOP);

1184 2     default:
1185 2          return (EP_RB_RECOVER_INVAL_OBJTYPE);
1186 1     }

1188 1     /* check if backup time must change */
1189 1     if ( backupTime != (time_t)0 )
1190 2     {
1191 2          /* get current backup time; if different from requested time,
1192 2           * then set requested time and save current time;
                                                          at end, restore
1193 2           * current time */

1195 2          if (rcp->rc_backup_app != 0)
1196 3          {
1197 3                  rtc = rcp-> currentPIptr-> piFuncArray[
                                                  PIFuncIndexGetCurTime ]
1198 3                                                      (
                                                       rcp, &save_time);
1199 3                  if (rtc == E_SUCCESS)
```

```
1200 3                          rtc = rcp-> currentPIptr->
1201 3                                      piFuncArray[
                                                 PIFuncIndexSetBkupTime ]
1202 3                                          (
                                        rcp, backupTime, backup_flags);
1203 3          } else {
1204 3                  rtc = RSTSL_GetCurrentBackupTime( &save_time );
1205 3                  if (rtc == E_SUCCESS)
1206 3                          rtc = RSTSL_SetBackupForTime (
                                            backupTime, backup_flags);
1207 2          }
1208 2          if (rtc != E_SUCCESS)
1209 2                  return rtc;
1210 1      }

1212 1      /* if this isn't a network backup restore,
                                            call plugin to do mark */
1213 1      if (rcp->rc_backup_app != 0)
1214 2      {
1215 2           rtc = rcp-> currentPIptr-> piFuncArray[ PIFuncIndexUnmark ]
1216 2                                  (
                                  rcp, thisObject, BadFilesOnly, descend,
1217 2                                 BadFilesCount, fileMarked,
1218 2                                 &lenMarkedFiles, dirMarked,
                                                          otherMarked,
1219 2                                 progressCB );

1221 2           /* save mark summary in restore_context */
1222 2           rcp->rc_mark_summary.nfiles = *fileMarked ;
1223 2           rcp->rc_mark_summary.ndirs = *dirMarked;
1224 2           rcp->rc_mark_summary.nother = *otherMarked;
1225 2           rcp->rc_mark_summary.len_mkd_files.high = lenMarkedFiles.high;
1226 2           rcp->rc_mark_summary.len_mkd_files.low = lenMarkedFiles.low;

1228 2           if (save_time)
1229 2                   rcp-> currentPIptr-> piFuncArray[
                                             PIFuncIndexSetBkupTime ]
1230 2                                          (
                                        rcp, save_time, backup_flags);


1232 2           return rtc;
1233 1      }

1235 1      *fileMarked = *dirMarked = *otherMarked = *BadFilesCount = 0;

1237 1      /*
1238 1       * The Below is static variable global to this module
                                                    api_markunmark.c
1239 1       *  They get initialized in EDMRST_UnmarkObject to 0.
1240 1       *  They get incremented in unmark_tree(
                                      ) when the badfiles are encountered.
1241 1       *  They get assigned to the output parameter to
                                            EDMRST_UnmarkObject after
1242 1       *  the unmark_tree call.
1243 1       */

1245 1      BADDATAFileUnmarkCount = 0;

1247 1      /*
1248 1       * input parameters
1249 1       * 1) BadfilesOnly --  is unmark limitted to Badfiles only
1250 1       * 2) descend --  does unmark descend into the contents of
                                                          directories.
1251 1       */

1253 1      if (BadFilesOnly)
```



```
1254 2      {
1255 2           mc.no_badfiles = TRUE;
1256 1      }
1257 1      else
1258 2      {
1259 2           mc.no_badfiles = FALSE;
1260 1      }

1262 1      if (!descend)
1263 2      {
1264 2           mc.no_descend = TRUE;
1265 1      }
1266 1      else
1267 2      {
1268 2           mc.no_descend = FALSE;;
1269 1      }

1271 1      /*
1272 1       * validate restorable object with the current mcat context
1273 1       */

1275 2      {
1276 2           tree_node *tnp_thisObject;
1277 2           int ret_unmark_tree;

1279 2           tnp_thisObject = mcat_lookup_path( rcp->rc_mcp,
1280 2                                              thisObject->root.objName );
1281 2           if (tnp_thisObject == NULL)
1282 3           {
1283 3                /*
1284 3                 *   restorable object must be corrupt mcat_lookup_path
                                                              could not
1285 3                 *   locate the tree node ptr for files
1286 3                 */

1288 3                return EP_RB_RECOVER_BAD_CONTEXT;
1289 2           }

1291 2           if (tnp_thisObject->tn_mcplane !=
1292 2                ((netBackupObjData *)(
                                  thisObject->appData.data))->objtnMCplane)
1293 3           {
1294 3                /*
1295 3                 * restorable object must be out of context return error
1296 3                 */

1298 3                return EP_RB_RECOVER_BAD_CONTEXT;
1299 2           }

1301 2           /*
1302 2            * If we made it here then thisObject is valid.
1303 2            * unmark_tree always returns 0
1304 2            */

1306 2           (void) unmark_tree(rcp, tnp_thisObject);
1307 1      }

1309 1      /*
1310 1       * The Below are static variable global to this module
                                                    api_markunmark.c
1311 1       *  They get initialized in EDMRST_UnmarkObject to 0.
1312 1       *  They get incremented in unmark_tree(
                                      ) when the badfiles are encountered.
1313 1       *  They get assigned to the output parameters to
                                            EDMRST_UnmarkObject after
1314 1       *  the unmark_tree call.
```

```
1315 1       */

1317 1      *BadFilesCount=BADDATAFileUnmarkCount;

1319 1      /*
1320 1       * if the summary is valid, set the total number of marked files,
1321 1       * directories, and "other" files. This is not intended to be the
1322 1       * number of files that resulted directly from the above
                                                          unmark_tree call.
1323 1       */

1325 1      if (rcp->rc_mark_summary_valid)
1326 2      {
1327 2           *fileMarked = rcp->rc_mark_summary.nfiles;
1328 2           *dirMarked = rcp->rc_mark_summary.ndirs;
1329 2           *otherMarked = rcp->rc_mark_summary.nother;
1330 1      }
1331 1      else
1332 2      {
1333 2           /*
1334 2            * This error is not fatal, the output variables *Marked,
1335 2            * will be zero, we should never get this error.
1336 2            */

1338 2           rbe_log_stats(
0, "Internal error: mark summary not Valid in RSTSL_UnmarkObject() ");
1339 1      }

1341 1      if (save_time)
1342 1           RSTSL_SetBackupForTime( save_time, backup_flags );

1344 1      return( E_SUCCESS );
1345 }         /* end of RSTSL_UnmarkObject () */
```



```
1347
1348      static void
          MarkUnmarkDebugLogEcho(tree_node *tnp)
1349 1    {
1350 1    #if defined(DEBUG_LOG_MARK_UNMK)
1351 2        {
1352 2             char pathbuf[EB_MAXPATHLEN];
1353 2             char *pbp = pathbuf;


1356 2             tn_getpath(tnp, &pbp);

1358 2             rbe_log_stats(0,"The file %s was marked", pbp);
1359 1        }
1360 1    #endif
1361 1    #if defined(DEBUG)
1362 2        {
1363 2             char pathbuf[EB_MAXPATHLEN];
1364 2             char *pbp = pathbuf;


1367 2             tn_getpath(tnp, &pbp);

1369 2             printf("The file %s was marked\n", pbp);
1370 1        }
1371 1    #endif
1372 1        return;
1373      }         /* MarkUnmarkDebugLogEcho */
```

```
2  /*****************************************************************************
                                                                          ******
3  **
4  ** File Name:   RSLsubmit.c
5  **
6  ** Copyright (c) 1998,1999 by EMC Corporation.
7  **
8  ** Purpose:
9  ** -----------
10 **       The intent of the contents of this file is to implement the
11 **       functions to create the submitObject and submitElement.
12 **
13 **       These functions are provided to allow:
14 **       - creation of submit objects,
                                  which define the set of objects to be
15 **         restored and the scripts to be run before and after
                                                          restoration,
16 **
17 ** The following functions comprise restoral management:
18 **
19 **       RSTSL_Submit
20 **
21 **
22 ** Compile-Time Options:
23 **               This section must list any compile time definitions
24 **               which will affect this header.
25 **
26 *****************************************************************************
                                                                       *****/



29 /*
30  * Feature test switches.
31  * Standard defines required to turn on OS features go here.
32  *
33  * The following is required for code that uses POSIX API's.
34  * Remove for non-POSIX, non-portable code.
35  */

37 #define  _POSIX_SOURCE 1
38 #define ULONG_TO_CHAR_SIZE 16


41 /*
42  * System headers.
43  */
44 /* for CreateSubmitname */
45 #include <string.h>
46 #include <sys/types.h>
47 #include <unistd.h>
48 /* for CreateSubmitname */
49 #include <sys/stat.h>
50 #include <fcntl.h>
51 #include <sys/wait.h>

53 /*
54  * Epoch headers.
55  */
56 #include <eb/eb_port.h>
57 #include <eb/rb_log.h>
58 #include <ebutil/eb_normalize.h>
59 #include <ebutil/ebutil.h>
60 #include <ebreport/ebvl.h>
61 #include <restore/RSLplugin.h>
```



```
63 /*
64  * Local headers
65  */

67 #include <RSLinterns.h>
68 #include <restore/EDMRESubmitApi.h>




73 void
74 fill_client_dirtop(struct recover_context *rcx);

76 static int
77 push_submit_file(struct recover_context *rcx,
78                  int fd,
79                  struct mark_summary *this_submit_files,
80                  ebvl_volidlist_ty **this_submit_volumes,
81                  struct mark_summary *total_submit_files,
82                  ebvl_volidlist_ty **total_submit_volumes,
83                  RSTSL_SubmitProgressProc progressCB,
84                  boolean_ty *submitCancelled);


87 static int
88 push_bfinfo_to_submitfile(struct recover_context *rcx,
89                           int plane,
90                           cat_descriptor *catd,
91                           long lmno,
92                           int fd,
93                           ebfs_uid_ty *prev_ebd,
94                           struct mark_summary *this_submit_files,
95                           ebvl_volidlist_ty **this_submit_volumes,
96                           struct mark_summary *total_submit_files,
97                           ebvl_volidlist_ty **total_submit_volumes);

99 static int
100 push_to_submitfile(int fd,
101                    char *buf,
102                    uint_t nbytes);

104 static void
105 ebfsid2str_lz(ebfs_uid_ty *ebfsidp,
106               register char *buf);

108 static int
109 ssID2ebfd(rbssID_t *ssidp,
110           ebfs_uid_ty *ebfdp);




115 /*****************************************************************************
116  * Restoral Management Functions:
117  *
118  *  These functions are provided to allow:
119  *       - creation of submit objects,
                                     which define the set of objects to be
120  *         restored and the scripts to be run before and after
                                                          restoration,
121  *       - starting the restoral of a submit object.
122  *
123  * The following functions comprise restoral management:
124  *
125  *       RSTSL_Submit
126  *       RSTSL_Start
```

```
 127    *
 128    ************************************************************************
 129    * Submit
 130    *
 131    * This function creates a submit object from the currently marked
 132    * restorable objects. It is passed to RSTSL_Start to begin execution
 133    * of the restore.
 134    *
 135    * Parameters:
 136    *
 137    * policy        (I) - The overwrite policy to use
 138    * inPlace       (
                  I) - Flag if the restoral is to be in original locations
 139    * hostName      (I) - host to restore to (only if inPlace == False)
 140    * directory     (I) - directory to restore to (
                                          only if inPlace == False)
 141    * transport      (
              I) - Indicator of transport the restoral is to be over (SCSI
 142    *                    or network)
 143    * submitObjIDptr(
                   IO) - ID of the submit user object created to describe
 144    *                    the restore
 145    * ObjectsSubmitted (O) - number of total file objects submitted.
 146    * progressCB    (
                  I) - pointer to callback function to report progress and
 147    *                    test for cancellation
 148    *
 149    ************************************************************************/
 150    eerrno_ty RSTSL_Submit(const char              *hostName,
 151                           const OverwritePolicy   policy,
 152                           const boolean_ty        inPlace,
 153                           const char              *directory,
 154                           const RestoreTransport  transport,
 155                           int                      *submitObjID,
 156                            unsigned int            *ObjectsSubmitted,
 157                            RSTSL_SubmitProgressProc progressCB,
 158                            EDMRST_submit_args       *submitArgs)
 159  1 {

 161  1   int submitElemID;
 162  1   int ret_status;
 163  1   int SEstatus = E_SUCCESS;
 164  1   int SOstatus = E_SUCCESS;
 165  1   int submit_fd;
 166  1   char submit_filename[2048];
 167  1   struct mark_summary *this_submit_files;
 168  1   ebvl_volidlist_ty *this_submit_volumes = NULL;
 169  1   struct mark_summary *total_submit_files;
 170  1   ebvl_volidlist_ty *total_submit_volumes = NULL;
 171  1   char *hostname_SE = NULL;
 172  1   char wi_type;
 173  1   boolean_ty submitCancelled = FALSE;
 174  1   char **envVar;
 175  1   int tmp;

 177  1   /*
 178  1    * The submitElement destructor should free these structures.
 179  1    */
 180  1   this_submit_files = malloc(sizeof(struct mark_summary));
 181  1   total_submit_files = malloc(sizeof(struct mark_summary));

 183  1   if((NULL == this_submit_files) ||
 184  1      (NULL ==total_submit_files))
 185  2   {
 186  2     rec_api_log_csm(SUB_CSM_NOMEM, NULL);
 187  2     rbe_log_stats(0, "Could not allocate memory RSTSL_Submit()");
```



```
 188  2     return(EP_RB_RECOVER_NOMEM);
 189  1   }

 191  1   memset(this_submit_files, 0, sizeof(struct mark_summary));
 192  1   memset(total_submit_files, 0, sizeof(struct mark_summary));



 196  1   /*
 197  1    * for Direct Connect, call its plugin version of Submit:
 198  1    * Did not change plug in call for CLI arguments for Port and Name
 199  1    */
 200  1   if (rcp->rc_backup_app != 0)
 201  1     return rcp-> currentPIptr-> piFuncArray[ PIFuncIndexSubmit ]
 202  1                             ( rcp,
 203  1                               hostName,
 204  1                               policy,
 205  1                               inPlace,
 206  1                               directory,
 207  1                               transport,
 208  1                               submitObjID,
 209  1                               progressCB );

 211  1   *submitObjID = NewSubmitObject(&SOstatus);

 213  1   if(E_SUCCESS != SOstatus)
 214  2     {
 215  2       rbe_log_stats(0, "Could not submit new object ID");
 216  2       return (EP_RB_RECOVER_FATALERR);
 217  1     }


 220  1   if (!inPlace && ((hostName == NULL) || ('\0' == hostName[0])))
 221  2     {
 222  2       rbe_log_stats(0, "Hostname or inplace not set");
 223  2       return(EP_RB_RECOVER_BAD_ARGS);
 224  1     }

 226  1   submitElemID = NewSubmitElement(*submitObjID, &SEstatus);

 228  1   if(E_SUCCESS != SEstatus)
 229  2     {
 230  2       rbe_log_stats(0, "Could not create new submit elements");
 231  2       return (EP_RB_RECOVER_FATALERR);
 232  1     }


 235  1   if(0 != SetSOBasics(*submitObjID,
 236  1                       0,
 237  1                       0,
 238  1                       &SOstatus))

 240  2     {
 241  2       rbe_log_stats(0, "Could not set SO basics");
 242  2       return (EP_RB_RECOVER_FATALERR);
 243  1     }

 245  1   /*
 246  1    * Set up the environment variables
 247  1    */
 248  1   envVar = calloc(2,sizeof(char *));
 249  1   envVar[0] = submitArgs->mapfile_env;
 250  1   if (NULL != envVar[0])
 251  2     {
 252  2       if (0 == strcmp(envVar[0], "\0"))
 253  3         {
```

```
254 3             envVar[0] = NULL;
255 2           }
256 1       }
257 1     envVar[1] = NULL;
258 1     if (NULL != envVar[0])
259 2       {
260 2         if (0 != SetSOExecutePhase(*submitObjID,NULL,NULL,envVar,&tmp))
261 3             {
262 3               rec_api_log_csm(FATAL_ERROR, NULL);
263 3               rbe_log_stats(
                      0, "Could not set the mapfile environment variable\n");
264 3               return(FATAL_ERROR);

266 2             }
267 1       }

269 1     /*
270 1      * Set the name of the client who initiated the call and the
271 1      * port to connect to.  Needs to be done before Direct connect call
272 1      */
273 1     if((0 != submitArgs->clientSocketPort) || (
                                        NULL != submitArgs->socketClientNm))
274 2       {
275 2         if ( 0 != SetSEEbciConnect(*submitObjID,
276 2                                    submitElemID,
277 2                                    submitArgs->socketClientNm,
278 2                                    submitArgs->clientSocketPort,
279 2                                    &SEstatus))
280 3             {
281 3               rbe_log_stats(
                            0, "Could not set socket port or client name");
282 3               return (EP_RB_RECOVER_FATALERR);
283 2             }
284 1       }

286 1       if(0 != SetSOAdminID(*submitObjID,
287 1                            (
              rcp->rc_recovery_flags & RC_RECFLAG_ADMINISTRATOR) ? 1 : 0,
288 1                            (
              rcp->rc_recovery_flags & RC_RECFLAG_SOURCE_SYSADMIN) ? 1 : 0,
289 1                            (
                 rcp->rc_recovery_flags & RC_RECFLAG_DEST_SYSADMIN) ? 1 : 0,
290 1                           &SOstatus))
291 2       {
292 2         return (EP_RB_RECOVER_FATALERR);
293 1       }

295 1       if ((NULL != rcp->rc_human_uidname) &&
296 1           (NULL != rcp->rc_effective_uidname))
297 2       {
298 2         if(0 != SetSOUserID(*submitObjID,
299 2                             rcp->rc_human_uid,
300 2                             rcp->rc_human_uidname,
301 2                             rcp->rc_effective_uid,
302 2                             rcp->rc_effective_uidname,
303 2                             &SOstatus))
304 3           {
305 3             return (EP_RB_RECOVER_FATALERR);
306 2           }

308 1       }
309 1       else
310 2       {
311 2         rbe_log_stats(
              0, "Restore context has not set human_name and/or effective_name,
                                                          in RSTSL_Submit()");
```



```
312 2         return(EP_RB_RECOVER_BAD_CONTEXT);
313 1       }

315 1       if(0 != SetSOVMCheck(*submitObjID,
316 1                            (
                          rcp->rc_recovery_flags & RC_RECFLAG_NO_VM_CHECK),
317 1                            &SOstatus))

319 2       {
320 2         return (EP_RB_RECOVER_FATALERR);
321 1       }

323 1       /*
324 1        * Initialize this at the start of each recover within a single
325 1        * recovery session.
                                This is important if the destination directory
326 1        * gets changed from a specific location to simply "in place".
327 1        */

329 1       if (NULL != rcp->rc_client_dirtop)
330 2       {
331 2           free(rcp->rc_client_dirtop);
332 1       }

334 1       rcp->rc_client_dirtop = NULL;

336 1       if (rcp->rc_client_hostname)
337 2       {
338 2           free(rcp->rc_client_hostname);
339 1       }

341 2       {
342 2         /*
343 2          * We need to get information about this work item from
344 2          * from the config structure. Currently this is just the
345 2          * work item type.
346 2          */
347 2         RBC_WORKGROUP *wgp;
348 2         boolean_ty wi_found = FALSE;

350 2         if(NULL == rcp->rc_config)
351 3         {
352 3           rbe_log_stats(
      0, "Restore context has not set up the config is RSTSL_Submit()");
353 3           return(EP_RB_RECOVER_BAD_CONTEXT);
354 2         }

356 2         if(NULL == rcp->rc_workitem_name)
357 3         {
358 3           rbe_log_stats(
  0, "Restore context has not set up current work item in RSTSL_Submit(
                                                                      )");
359 3           return(EP_RB_RECOVER_BAD_CONTEXT);
360 2         }

362 2         wi_found = GetClientTypeFromConfig(rcp->rc_config,
363 2                                            rcp->rc_workitem_name,
364 2                                            &wi_type);
365 2         if (FALSE == wi_found)
366 3         {
367 3           rbe_log_stats(
                0, "Could not find the work item in the config structure in
                                                              RSTSL_Submit()");
368 3           return(EP_RB_RECOVER_BAD_CONTEXT);
369 2         }
370 1       }
```

```
371 1     if (inPlace)
372 2     {
373 2       if(NULL == rcp->rc_source_client_hostname)
374 3       {
375 3          return(EP_RB_RECOVER_BAD_CONTEXT);
376 2       }
377 2       else
378 3       {
379 3          hostname_SE = esl_strdup(rcp->rc_source_client_hostname);
380 2       }

382 1     }
383 1     else /* !inPlace */
384 2     {
385 2       /* hostName is checked above if !inPlace */

387 2       hostname_SE = (char *) hostName;
388 1     }
389 1     if (NULL == (rcp->rc_client_hostname = esl_strdup((
                                                    char *)hostname_SE)))
390 2     {
391 2          rec_api_log_csm(SUB_CSM_NOMEM, NULL);
392 2          rbe_log_stats(0, "Could not allocate memory RSTSL_Submit()");
393 2          return(EP_RB_RECOVER_NOMEM);
394 1     }

396 1     rcp->rc_overwrite_policy = policy;

398 1     /*
399 1      * fill_client_dirtop must always be called because
400 1      * cross recoveries for NOS clients require some special
401 1      * handling. The destination server name needs to be
402 1      * prepended along with a : in order for it to build the
403 1      * correct target string on the recover command sent to the client.
404 1      * fill_client_dirtop() handles this correctly.
405 1      */

407 1     if(0 != fill_client_dirtop2(rcp->rc_config,
408 1                                 inPlace,
409 1                                 (!inPlace) ? (char *) directory : (
                                                        char *) NULL,
410 1                                 rcp->rc_workitem_name,
411 1                                 hostname_SE,
412 1                                 &(rcp->rc_client_dirtop)))
413 2     {
414 2          rbe_log_stats(0, "Internal error:"\
415 2                        "Could not file client dirtop in RSTSL_Submit(
                                                              )");

417 2          return(EP_RB_RECOVER_BAD_CONTEXT);
418 1     }

420 1     if (NULL == rcp->rc_client_dirtop)
421 2     {
422 2          rec_api_log_csm(SUB_CSM_NOMEM, NULL);
423 2          rbe_log_stats(0, "Could not allocate memory RSTSL_Submit()");
424 2          return(EP_RB_RECOVER_NOMEM);
425 1     }

427 1     if((NULL != rcp->rc_workitem_name) &&
428 1        (NULL != rcp->rc_template_name) &&
429 1        (NULL != rcp->rc_source_client_hostname))
430 2     {
431 2       if ( 0 != SetSEBasics(*submitObjID,
432 2                             submitElemID,
433 2                             rcp->rc_workitem_name,
```



```
434 2                             rcp->rc_template_name,
435 2                             rcp->rc_saveset_thread,
436 2                             rcp->rc_source_client_hostname,
437 2                             wi_type,
438 2                             &SEstatus))

440 3       {
441 3          return (EP_RB_RECOVER_FATALERR);
442 2       }
443 1     }
444 1     else
445 2     {
446 2       rbe_log_stats(
                          0, "Restore context has not set template_name,
              witem_name and/or source_client_name, in RSTSL_Submit()");
447 2       return(EP_RB_RECOVER_BAD_CONTEXT);
448 1     }



452 1     /* directory & hostName are check as input args */
453 1     /* What about transport ?? */

455 1     if(0 != SetSEDestination(*submitObjID,
456 1                              submitElemID,
457 1                              (char *)hostname_SE,
458 1                              inPlace,
459 1                              (char *)directory,
460 1                              policy,
461 1                              transport,
462 1                              &SEstatus))
463 2     {
464 2       return (EP_RB_RECOVER_FATALERR);
465 1     }

467 1     if(NULL != rcp->rc_client_dirtop)
468 2     {
469 2       if(0 != SetSEDirtop(*submitObjID,
470 2                           submitElemID,
471 2                           rcp->rc_client_dirtop,
472 2                           &SEstatus))
473 3       {
474 3          return (EP_RB_RECOVER_FATALERR);
475 2       }
476 1     }
477 1     else
478 2     {
479 2       rbe_log_stats(
              0, "Restore context has not set dirtop, in RSTSL_Submit()");
480 2       return(EP_RB_RECOVER_BAD_CONTEXT);
481 1     }

483 1     if((NULL != rcp->rc_client_scriptname) &&
484 1        (NULL != rcp->rc_client_rbuname))
485 2     {
486 2       if(0 != SetSEScriptName(*submitObjID,
487 2                               submitElemID,
488 2                               rcp->rc_client_scriptname,
489 2                               rcp->rc_client_rbuname,
490 2                               &SEstatus))
491 3       {
492 3          return (EP_RB_RECOVER_FATALERR);
493 2       }
494 1     }

496 1     else
```

```
497  2      {
498  2        rbe_log_stats(
                0, "Restore context has not set scriptname or client user_name,
                                                        in RSTSL_Submit()");
499  2        return(EP_RB_RECOVER_BAD_CONTEXT);
500  1      }

502  1      /*
503  1       * Progress callback intended to test for cancelation and
504  1       * to report progress on the submit to the user.
505  1       */
506  1      if(TRUE == progressCB(0))
507  2      {
508  2        /* Lets clean up here!
509  2         * right now there is no clean up routine for submitObjects.
510  2         */
511  2        rbe_log_stats(EP_RB_RECOVER_ABORT,"User abort during submit.");
512  2        submitCancelled = TRUE;
513  2        return(EP_RB_RECOVER_ABORT);
514  1      }
515  1      submit_fd = OpenSubmitFile(TRUE,
516  1                                 *submitObjID,
517  1                                 submitElemID,
518  1                                 &SEstatus);


521  1      if(-1 == submit_fd)
522  2      {
523  2        return (EP_RB_RECOVER_FATALERR);
524  1      }

526  1      ret_status = push_submit_file(rcp,
527  1                                    submit_fd,
528  1                                    this_submit_files,
529  1                                    &this_submit_volumes,
530  1                                    total_submit_files,
531  1                                    &total_submit_volumes,
532  1                                    progressCB,
533  1                                    &submitCancelled);


536  1      CloseSubmitFile(submit_fd, TRUE, &SEstatus);

538  1      if(TRUE == submitCancelled)
539  2      {
540  2        /* Lets clean up here!
541  2         * right now there is no clean up routine for submitObjects.
542  2         */
543  2        rbe_log_stats(EP_RB_RECOVER_ABORT,"User abort during submit.");
544  2        return(EP_RB_RECOVER_ABORT);
545  1      }

547  1      if(-1 == ret_status)
548  2      {
549  2        return (EP_RB_RECOVER_FATALERR);
550  1      }

552  1      *ObjectsSubmitted = (unsigned int) ret_status;

554  1      if(0 != SetSOTotalSize(*submitObjID,
555  1                             total_submit_files,
556  1                             &SOstatus))
557  2      {
558  2        ; /* Not sure this one needs to be handled */
559  1      }
```



```
561  1      if(0 !=SetSOTotalVolumes(*submitObjID,
562  1                               total_submit_volumes,
563  1                               &SOstatus))
564  2      {
565  2        ; /* Not sure this one needs to be handled */
566  1      }

568  1      if(0 != SetSESummary(*submitObjID, submitElemID,
569  1                           this_submit_files, &SEstatus))
570  2      {
571  2        ; /* Not sure this one needs to be handled */
572  1      }
573  1      if(0 != SetSEVolumes(*submitObjID,
574  1                           submitElemID,
575  1                           this_submit_volumes,
576  1                           &SEstatus))
577  2      {
578  2        ; /* Not sure this one needs to be handled */
579  1      }

581  1      return( E_SUCCESS );
582    }
```

```
586   void
587   fill_client_dirtop(struct recover_context *rcx)
588 1 {
589 1     char buf[4096];
590 1     RBC_WORKITEM *pwi;
591 1     RBC_WORKGROUP *pwg;
592 1     boolean_ty cross_recover;


595 1     for (pwg = rcx->rc_config->pgrouplist; NULL != pwg;
                                                        pwg = pwg->next)
596 2     {
597 2          for (pwi = pwg->pwilist; NULL != pwi; pwi = pwi->next)
598 3          {
599 3              if (0 == strcmp(pwi->name, rcx->rc_workitem_name))
600 4              {
601 4                  goto gotit2;
602 3              }
603 2          }
604 1     }

606 1 gotit2:

608 1     /*
609 1      * if the dirtop already specifies a netware client
610 1      * target, remove it first.
611 1      */

613 1     if (rcx->rc_client_dirtop != NULL
614 1          && NULL != strchr(rcx->rc_client_dirtop, ':'))
615 2     {
616 2          return;
617 1     }

619 1     /*
620 1      * cross_recover is a boolean variable used to indicate a
                                                    cross-recover request.
621 1      * This will set the proper target for NOS clients and should NOT
                                                        affect others.
622 1      */

624 1     cross_recover = (NULL != rcx->rc_client_hostname) &&
625 1                 (0 != strcmp(
              rcx->rc_source_client_hostname, rcx->rc_client_hostname));

627 1     sprintf(buf, "%s%s%s",
628 1            ((NULL != pwi && NULL != pwi->nw_clnt_target)
629 1                ? (cross_recover
630 1                    ? rcx->rc_client_hostname
631 1                    : pwi->nw_clnt_target)
632 1                : ""),
633 1             ((NULL != pwi && NULL != pwi->nw_clnt_target) ? ":" : ""),
634 1             ((
        NULL != rcx->rc_client_dirtop) ? rcx->rc_client_dirtop : "/"));

636 1     if (rcx->rc_client_dirtop != NULL)
637 2     {
638 2          free (rcx->rc_client_dirtop);
639 1     }

641 1     rcx->rc_client_dirtop = esl_strdup(buf);
642 1     if (NULL == rcx->rc_client_dirtop)
643 2     {
```



```
644 2          rec_api_log_csm(SUB_CSM_NOMEM, NULL);
645 1     }
646   }          /* fill_client_dirtop */
```

```
651  static int
652  push_submit_file(struct recover_context *rcx,
653                   int submit_fd,
654                   struct mark_summary *this_submit_files,
655                   ebvl_volidlist_ty **this_submit_volumes,
656                   struct mark_summary *total_submit_files,
657                   ebvl_volidlist_ty **total_submit_volumes,
658                   RSTSL_SubmitProgressProc progressCB,
659                   boolean_ty *submitCancelled)
660 1 {

662 1   int submit_cont = 1;
663 1   int retStatus = 0;
664 1   int bitfiles_pushed = 0;
665 1   ebfs_uid_ty prev_ebd;

667 1   if(NULL != submitCancelled)
668 1     *submitCancelled = FALSE;
669 1   else
670 1     return -1;

672 1   memset(&prev_ebd, 0, sizeof(ebfs_uid_ty));


675 1   if((NULL == rcx) ||
676 1      (NULL == this_submit_files) ||
677 1      (NULL == this_submit_volumes) ||
678 1      (NULL == total_submit_files) ||
679 1      (NULL == total_submit_volumes))
680 2   {
681 2     return -1;
682 1   }

684 1   if (submit_cont)
685 2   {

687 2         int plane;

689 2        /* MCAT_TOP is #define in mcat.h */
690 2         for (plane = (-rcx->rc_nplanes)+1; plane <= MCAT_TOP;
                                                          plane++)
691 3         {
692 3             cat_descriptor *catd = mcat_getcatd(rcx->rc_mcp, plane);
693 3             char *marks = rcx->rc_marks[-plane];
694 3             long ntrlm;
695 3             long lmno;


698 3             if (catd == NULL)   /* should never happen */
699 4             {
700 4                 continue;
701 3             }
702 3             /* catdesc.h */
703 3             ntrlm = catd_ntrlm(catd);

705 3             for (lmno = 0; lmno < ntrlm; lmno++)
706 4             {
707 4               /* RSLrbrmain.h */
708 4                 if (! TLMNO_MARKED(marks, lmno))
709 5                 {
710 5                     continue;
711 4                 }
```



```
713 4                   retStatus = push_bfinfo_to_submitfile(rcx,
714 4                                                         plane, catd,
                                                                  lmno,
715 4                                                         submit_fd,
716 4                                                         &prev_ebd,
717 4
                                                            this_submit_files,
718 4
                                                          this_submit_volumes,
719 4
                                                           total_submit_files,
720 4
                                                         total_submit_volumes);
721 4                   if (1 == retStatus)
722 5                   {
723 5                     bitfiles_pushed++;
724 4                   }
725 4                   if(bitfiles_pushed % 1024)
726 5                   {
727 5                     if(TRUE == progressCB(bitfiles_pushed))
728 6                     {

730 6                       rbe_log_stats(EP_RB_RECOVER_ABORT,
731 6                                     "User abort during submit.");
732 6                       *submitCancelled = TRUE;
733 6                       return(-1);
734 5                     }

736 4                   }
737 3               }
738 2           }

740 2           if(-1 == retStatus)
741 3           {
742 3             return -1;
743 2           }
744 2           return bitfiles_pushed;
745 1       }
746   }
```

```
749   /*
750    * Returns: -1 for file not submitted for restore, error encountered.
751    *           0 for file not submitted for restore,
                                                    NO error encountered.
752    *           1 for file subitted successfully.
753    */
754   static int
755   push_bfinfo_to_submitfile(struct recover_context *rcx,
756                             int plane,
757                             cat_descriptor *catd,
758                             long lmno,
759                             int fd,
760                             ebfs_uid_ty *prev_ebd,
761                             struct mark_summary *this_submit_files,
762                             ebvl_volidlist_ty **this_submit_volumes,
763                             struct mark_summary *total_submit_files,
764                             ebvl_volidlist_ty **total_submit_volumes)
765 1 {
766 1   /* Add mark support */
767 1     char              ebfsbfstr[34];
768 1     rbtree_elem_t     tlm;
769 1     rbcat_elem_t      clm;
770 1     long              catlmno;
771 1     cat_descriptor  *catlm_catd;
772 1     char              ebfsdirstr[34];
773 1     char              buf[100];
774 1     size_t            nbytes;
775 1     size_t            namesize = 0;
776 1     char             *fname = "<name unknown>";
777 1     char              *this_file;
778 1     eperrno            ep_status;
779 1     ebfs_uid_ty ebd;
780 1     ebfs_uid_ty zero_bitfileid;
781 1     char              *directives_list=NULL;
782 1     int               directives_size=0;


785 1     (void)catd_read_trlm(catd, lmno, &tlm, &this_file, (size_t *)NULL);

787 1     /*
788 1      * Get the corresponding catalog element.
789 1      * Note that it might come from a different
790 1      * plane if this tree element is a DS_NONE.
791 1      */

793 1     if (tlm.te_catelem != -1)
794 2     {
795 2          /*
796 2           * not DS_NONE -- the common case
797 2           */

799 2          catlmno = tlm.te_catelem;
800 2          catlm_catd = catd;
801 1     }
802 1     else
803 2     {
804 2          int clmplane;

806 2          /*
807 2           * This is a DS_NONE record.  Get
808 2           * the real corresponding catalog element.
809 2           */

811 2          dsnone_get_realcat(rcx, lmno, plane, &catlmno, &clmplane);
```



```
813 2          /*
814 2           * There is a special case for the root ("/")
815 2           * in the backup catalogs.  It's in the tree file
816 2           * but not in any catalog file.  This case can also
817 2           * occur for leading directories that are "above"
818 2           * the starting point for a work-item.  Silently
819 2           * ignore such directories, unless debugmode is on.
820 2           */

822 2          if (catlmno == -1)
823 3          {
824 3              size_t len;

826 3              (void)catd_read_trlm(catd, lmno, &tlm, &fname, &len);
827 3              if (len != 0 && debugmode) /* len 0 filters out "/" */
828 4              {
829 4                /*rbe_log_stats(
                   0, "** warning: cannot find catalog record for file %s;"
830 4                        " skipping it.", fname);*/
831 3              }
832 3              return 0;
833 2          }
834 2          catlm_catd = mcat_getcatd(rcx->rc_mcp, clmplane);
835 1     }
836 1     (void)catd_read_catlm(catlm_catd, catlmno, &clm, (char **)NULL);

838 1     add_to_summary(this_submit_files, &clm);
839 1     add_to_summary(total_submit_files, &clm);

841 1     *this_submit_volumes = ebvl_genvolidlist(*this_submit_volumes,
842 1                                              &clm.ce_bitfileID,
843 1                                              1,
                                                 ebvl_EbfsIdType_BitFile,
844 1                                              &ep_status);

846 1     if((NULL == *this_submit_volumes) || (0 != ep_status))
847 2     {
848 2       rbe_log_stats(
                 0, "** warning: cannot maintain volume list for submit."
849 2                      " skipping it.");
850 1     }

852 1     *total_submit_volumes = ebvl_genvolidlist(*total_submit_volumes,
853 1                                               &clm.ce_bitfileID,
854 1                                               1,
                                                  ebvl_EbfsIdType_BitFile,
855 1                                               &ep_status);

857 1     if((NULL == *total_submit_volumes) || (0 != ep_status))
858 2     {
859 2       rbe_log_stats(
                 0, "** warning: cannot maintain volume list for submit."
860 2                      " skipping it.");
861 1     }


864 1     memset(&zero_bitfileid, 0, sizeof(ebfs_uid_ty));


867 1     if(0 == memcmp(&zero_bitfileid,
868 1                    &clm.ce_bitfileID,
869 1                    sizeof(ebfs_uid_ty)))
870 2     {
871 2       (void)catd_read_catlm(catlm_catd, catlmno, &clm, &fname);
```

```
873 2       rbe_log_stats(
                    0, "** warning: There is no backup data to recover for "
874 2                  "file \"%s\"; skipping it.", fname);
875 2       return 0;
876 1     }


879 1     /*
880 1      * If this is a renamed element,
                                        must get the current name from cat
881 1      */


884 1     (void)catd_read_catlm(catlm_catd, catlmno, &clm, &fname);
885 1     if (clm.ce_status & CESTAT_RENAME)
886 2       {
887 2          /*
888 2           * name may contains substrings as in netware
889 2           */
890 2          namesize = (size_t)clm.ce_namelen;
891 1       }

893 1     if(0 != ssID2ebfd(&clm.ce_ssID, &ebd))
894 2     {
895 2       rbe_log_stats(
               0, "** warning: could not determine bitfile directory for "
896 2                  "file \"%s\"; skipping it.", fname);
897 2       return 0;
898 1     }

900 1     if ((NULL != rcx->recx_directives_p) || (NULL != fname))
901 2       {
902 2          directives_list=RSTSL_get_directives_for_file(
                                                  rcx->recx_directives_p,
903 2                                                        fname);
904 1       }
905 1     else
906 2       {
907 2          directives_list = NULL;
908 1       }
909 1     /* done in case string not null terminated */
910 1     if(directives_list !=NULL)
911 2       {
912 2          directives_size = strlen(directives_list);
913 1       }
914 1     else
915 2       {
916 2          directives_size = 0;
917 1       }

919 1     if(0 != WriteBitfileInfoToSubmitFile(fd,
920 1                                          &ebd,
921 1                                          &clm.ce_bitfileID,
922 1                                          clm.ce_mode,
923 1                                          namesize,
924 1                                          (
                                            namesize > 0) ? fname: NULL,
925 1                                          directives_size,
                                                  /* directive_size */
926 1                                          directives_list,
                                                     /* directives */
927 1                                          clm.ce_filesize,
928 1                                          prev_ebd))
929 2     {
930 2       rbe_log_stats(
                0, "** warning: could not submit marked file for restore."
```



```
931 2                                  " skipping file %s.", fname);
932 2       return 0;
933 1     }

935 1     memcpy(prev_ebd, &ebd, sizeof(ebfs_uid_ty));

937 1   #if 0     /* G.
                  Sachar: since buf is uninitialized and function is #if 0'd */
938 1     push_to_submitfile(fd, buf, strlen(buf));
939 1   #endif

941 1     return 1;

943 1     /*
944 1      * Creation and destruction of volid's now taks place
945 1      * outside of "go" command.
946 1      *
947 1      * now add bitfile id to volumes needed report
948 1      * if (rcx->ebvlok)
949 1      * {
950 1      *     rcx->ebvllist = ebvl_genvolidlist(
                                              rcx->ebvllist, &clm.ce_bitfileID,
951 1      *                                             1,
                                               ebvl_EbfsIdType_BitFile,
952 1      *                                            &ep_status);
953 1      * }
954 1      * if (rcx->ebvlok && NULL == rcx->ebvllist)
955 1      * {
956 1      *     fprintf(
                   stderr, "Unable to generate volumes needed report, %s (%d)",
957 1      *      e_get_error_text(ep_status), ep_status);
958 1      *      rcx->ebvlok = FALSE;
959 1      * }
960 1      */
961     }         /* end of push_bfinfo_to_submitfile() */
```

```
964     /* Returns: -1 for error encountered
965      *          otherwise number of bytes written
966      *
967      *
968      *
969      */

971     static int
972     push_to_submitfile(int fd,
973                        char *buf,
974                        uint_t nbytes)
975   1 {
976   1 #if 0
977   1     int wrote;
978   1     int save_errno;

980   1     if (debugmode)
981   2     {
982   2          (void)write(fileno(stdout), buf, nbytes);
983   1     }

985   1     wrote = looprw(fd, buf, (int)nbytes, write_no_eintr);
986   1     save_errno = errno;
987   1     if (wrote != (int)nbytes)
988   2     {
989   2       /*
990   2        * short write error
991   2        */
992   2       rbe_log_stats(0, "\n** Trouble writing submitfile.");
993   2       rbe_log_stats(
                    0, "** wanted to write %d, wrote %d", nbytes, wrote);

995   1     }
996   1     errno = save_errno;
997   1     return wrote;
998   1 #endif
999   1     return nbytes;
1000    }         /* end of push_to_submitfile() */
```



```
1002    /*
1003     * Convert an EBFS ID to string form.  Elide leading zeros.
1004     */

1006    static void
1007    ebfsid2str_lz(ebfs_uid_ty *ebfsidp,
1008                  register char *buf)
1009  1 {
1010  1     register char *p;
1011  1     register char *q;
1012  1     unsigned long longvals[4];
1013  1     int i;
1014  1     char tmpbuf[ 33 ];
1015  1     char *hexdigits = "0123456789abcdef";


1018  1     memcpy(longvals, ebfsidp, 16);

1020  1     q = tmpbuf;

1022  1     for (i = 0; i < 4; i++)
1023  2     {
1024  2          int j;
1025  2          register unsigned long ul;

1027  2          q += 8;
1028  2          ul = longvals[i];

1030  2          for (j = 0; j < 8; j++)
1031  3          {
1032  3              *--q = hexdigits[ LONG2INT(ul & 0xF) ];
1033  3              ul >>= 4;
1034  2          }
1035  2          q += 8;
1036  1     }

1038  1     tmpbuf[32] = '\0';

1040  1     /*
1041  1      * Skip over leading 0 characters
1042  1      */

1044  1     for (q = tmpbuf; *q == '0'; q++)
1045  2     {
1046  2          /* null */
1047  1     }

1049  1     /*
1050  1      * Copy the rest, up to and including the comma, to output buf
1051  1      */

1053  1     p = buf;
1054  1     while ((*p++ = *q++) != '\0')
1055  2     {
1056  2          /* null */
1057  1     }
1058    }         /* end of ebfsid2str_lz() */
```

```
1061    static int
1062    ssID2ebfd(rbssID_t *ssidp,
1063              ebfs_uid_ty *ebfdp)
1064  1 {
1065  1     rbsaveset_t           ss;
1066  1     eperrno               err;


1069  1     /*
1070  1      * clobber it, to make failure to find match obvious
1071  1      */

1073  1    (void)memset((char *)ebfdp, 0, sizeof *ebfdp);


1076  1     if ((err = ss_find1(ssidp, &ss, 0)) == 0)
1077  2     {
1078  2       memcpy(ebfdp, &ss.ss_dirID, sizeof(ebfs_uid_ty));
1079  1     }

1081  1     return err ? -1 : 0;
1082    }       /* end of ssID2ebfd() */
```



```
1086    /*****************************************************************
                                                                        ********
1087    **
1088    ** Routine:  RSTSL_get_catalog_info
1089    **
1090    ** Inputs:  time       - The time of the backup that is being worked
                                                                             with
1091    **
1092    ** Outputs:  level     - reference to the level string to be output
1093    **                      will contain the level of the backup
1094    **           numrec     - reference to the string which will contain the
                                                                           number
1095    **                      of records for the backup
1096    **           catType    - reference to the string which will contain the
                                                                           type
1097    **                      of catalog for the specified backup
1098    **
1099    ** Purpose:  Function to retrieve backup level, catalog type,
                                                                  and number of
1100    **           records and then retun it to the client
1101    **
1102    ** Return Codes: E_SUCCESS   - if the cattalog exists and able to
                                                                        return
1103    **                         information
1104    **               EP_RB_RECOVER_NO_CATALOG  - error getting the catalog
                                                                         info
1105    **
1106    *****************************************************************
                                                                        *******
1107    */
1108    eerrno_ty
1109    RSTSL_get_catalog_info( const time_t time,
1110                            char **level,
1111                            char **numrec,
1112                            char **catType)
1113         /* Called from re_get_catalog_info_1_svc RPC call*/
1114  1 {
1115  1       cat_descriptor *catdPtr;
                                    /* pointer to the catalog descriptor */
1116  1       rbcat_head_t cathdr;
                                /* pointer to head of a catalog retrieved from
1117  1                            * the catalog descriptor
1118  1                            */
1119  1       int plane_count=0;
                                /* used to keep track of the traversal through
1120  1                            * the catalog plane structure
1121  1                            */
1122  1       int number_of_planes;
                                   /* total number of planes to traverse */


1125  1       number_of_planes=-(rcp->rc_nplanes);
                                             /* must be negated because of
1126  1                                         * previous
                                                    implementation of
1127  1                                         * cat struct
1128  1                                         */

1130  1       do  /* traverse the list of planes in the catalog structure */
1131  2          {
1132  2            /* get a pointer to the plane I want staring at 0
1133  2             * and counting down to the number of planes in catalog
                                                                        struct
```

```
1134  2            */
1135  2          catdPtr = mcat_getcatd(rcp->rc_mcp, plane_count);
                                                    /* get the head */
1136  2          if (NULL == catdPtr)  /* if there are no catalogs */
1137  3            {
1138  3              return(EP_RB_RECOVER_NO_CATALOG);
1139  2            }

1141  2          if (0 != catd_get_cat_head(
                           catdPtr, &cathdr))  /* if there is no head
1142  2                                           * there is
                                                    still not
1143  2                                          *catalog
1144  2                                          */
1145  3            {
1146  3              return(EP_RB_RECOVER_NO_CATALOG);
1147  2            }

1149  2          plane_count--;
                        /* decrement counter, must go into negatives */

1151  2          /* while the backup we are looking for has not been found,
                                                                    and
1152  2           * we have not traversed through the entire list
1153  2           */
1154  1          }while ((time !=cathdr.ch_time)&&(
                                       plane_count>=number_of_planes));

1156  1        *numrec=(char *)malloc( ULONG_TO_CHAR_SIZE );
                                                  /* must malloc memory
1157  1                                             * because cannot
                                                       strdup
1158  1                                             * a ulong
1159  1                                             */
1160  1        sprintf(*numrec,"%u",cathdr.ch_nelem);
                                         /* make the ulong a string
1161  1                                             * which is copied
                                                      from the
1162  1                                             *head of the
                                                       catalog
1163  1                                             */
1164  1        (*level) = (char *)calloc(1,2);
1165  1        (*level)[0] = cathdr.ch_level;

1167  1        switch (cathdr.ch_state)
1168  2           {
1169  2           case SSCAT_PARTIAL:
1170  2                *catType=esl_strdup("PARTIAL");
1171  2                break;
1172  2           case SSCAT_UNSORTED:
1173  2                *catType=esl_strdup("UNSORTED");
1174  2                break;
1175  2           case SSCAT_SORTED:
1176  2                *catType=esl_strdup("SORTED");
1177  2                break;
1178  2           case SSCAT_FULL:
1179  2                *catType=esl_strdup("FULL");
1180  2                break;
1181  2           case SSCAT_DELTA:
1182  2                *catType=esl_strdup("DELTA");
1183  2                break;
1184  2           case SSCAT_EXPIRED:
1185  2                *catType=esl_strdup("EXPIRED");
1186  2                break;
1187  2           case SSCAT_NONE:
1188  2                *catType=esl_strdup("NONE");
```



```
1189  2                break;
1190  2            default:
1191  2                *catType=esl_strdup("UNKNOWN");
1192  2                break;
1193  1           }

1195  1         return E_SUCCESS;
1196      }
```

```
#ifdef TESTMAIN
#define LONG2INT(x)       (x)
#endif

#define __EXTENSIONS__          /* needed by eb headers: ebutil ? */

#include <stdio.h>
#include <string.h>
extern char *strdup(const char *);
#include <stdlib.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <limits.h>

#include <esl/c_portable.h>
#include <util/hyper.h>
#include <ebfs/ebfs_if.h>

#include <restore/EDMRESubmitApi.h>

#include "SubmitFile.h"

extern int
GetSESubmitFile(int ID, int elementID, char *buff, int maxsize,
                int *status);
extern int
SetSESubmitFile(int ID, int elementID, char *submit_file, int *status);


#ifdef TESTMAIN
struct submitfile_bitfile_info
{
  char ebfs_dirp[33];
  char ebfs_filep[33];
  boolean_ty is_bfile_dir;
  int file_renamed_size; /*Needed for Netware */
  char *file_renamed; /*NULL if no rename*/
  int directive_size;
  char *directives;
  ulong filesize_high;
  ulong filesize_low;
};

char *testmain_filename_G = NULL;

#endif

/*
 * array to convert ascii hex character to hex value
 */
uchar_t hdigit_conv[256] =
{

     /*
      * ascii characters begin here
      */

     0, 0, 0, 0, 0, 0, 0, 0,   0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0,   0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0,   0, 0, 0, 0, 0, 0, 0, 0,
     0, 1, 2, 3, 4, 5, 6, 7,   8, 9, 0, 0, 0, 0, 0, 0,
     0,10,11,12,13,14,15, 0,   0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0,   0, 0, 0, 0, 0, 0, 0, 0,
     0,10,11,12,13,14,15, 0,   0, 0, 0, 0, 0, 0, 0, 0,
```



```
     0, 0, 0, 0, 0, 0, 0, 0,   0, 0, 0, 0, 0, 0, 0, 0,

     0, 0, 0, 0, 0, 0, 0, 0,   0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0,   0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0,   0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0,   0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0,   0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0,   0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0,   0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0,   0, 0, 0, 0, 0, 0, 0, 0,
};


static int
WriteTrailorToSubmitFile(const int submit_fd,
                         int *status);

static void
ebfsid2str_lz(const ebfs_uid_ty *ebfsidp,
              register char *buf);

int
str_lz2ebfsid(char *ebfsdirstr, ebfs_uid_ty *ebfs_p);

static ssize_t
Write(int fildes, const void *buf, size_t nbyte);

static ssize_t
Read(int fildes, void *buf, size_t nbyte);

/* What about ebfsid2str_lz ?? */

static char
*CreateSubmitfileName(int SubmitObjectID,
                      int SubmitElemID,
                      int *status)

{
    int buffersize = 0;
    int fd;
    int counter = 0;
    boolean_ty done_flag = FALSE;
    char buffer[E_PATH_MAX];
    char *submit_file_name = NULL;
    buffer[0] = 0;


    sprintf(
         buffer, "%s/%s.%d.%d", SUBMIT_FILE_DIR, SUBMIT_FILE_BASENAME,
             getpid(), SubmitElemID);

    while (done_flag == FALSE)
    {
        if (-1 == access(buffer, F_OK))
        {
            if (ENOENT == errno)
            {
                done_flag = TRUE;
            }
            else if (EINTR == errno)
            {
                continue;
            }
        }
        else
            sprintf(buffer, "%s/%s.%d.%d.%d",
```

```
132 2                       SUBMIT_FILE_DIR, SUBMIT_FILE_BASENAME,
133 2                       getpid(), SubmitElemID, counter++);
134 1     }



137 1     while (1)
138 2     {
139 2          fd = open(buffer, O_WRONLY|O_CREAT|O_TRUNC, S_IRUSR|S_IWUSR);

141 2          if (fd == -1)
142 3          {
143 3               if (errno == EINTR)
144 4               {
145 4                    continue;
146 3               }
147 3               else
148 4               {
149 4                    *status = errno;
150 4                    return NULL;
151 3               }
152 2          }

154 2          break;
155 1     }

157 1     close(fd);

159 1     submit_file_name = (char *) strdup(buffer);

161 1     if (submit_file_name == NULL)
162 2     {
163 2          *status = ENOMEM;
164 2           return NULL;
165 1     }

167 1     return submit_file_name;
168  }
```



```
170  /*
171   * OpenSubmitFile()
172   *  To be called to open a submit file. The submitfile
173   *  name will set if thie is open for write. If open for
174   *  read the submit file be retrieved in the submitElement.
175   *  If opened for write the file will be created with Write
176   *  and Read for the owner (which should be root) and
177   *  opened for writing.
178   *  If open for read the mod will be set to be read by the
179   *  owner only and the file opened for read only.
180   *
181   * Args:
182   *  (I) boolean_ty WriteAccess -- TRUE if opening for write.
                                                    FALSE for read.
183   *  (I) int ID  -- for Submit Object ID.
184   *  (I) int elementID -- Submit Element ID.
185   *  (O) int *status -- errno that caused failure.
186   *
187   * Returns: int
188   *  Less than zero for failure.
189   *  zero or greater is the fd of the open submit file.
190   *
191   */
192  int OpenSubmitFile(boolean_ty WriteAccess,
193                     int ID,
194                     int elementID,
195                     int *status)
196 1 {
197 1   int fd;
198 1   char *SubmitFileName;
199 1   int oflag;
200 1   mode_t mode;
201 1   int CreateStatus;
202 1   int SetStatus;
203 1   int SubmitObjectID = ID;
204 1   int SubmitElemID = elementID;

206 1   char SubmitFileBuf[E_PATH_MAX];

208 1   /* Get the submit file name */

210 1   if(TRUE == WriteAccess)
211 2   {
212 2     int CreateStatus;
213 2     int SetStatus;

215 2     if(NULL == (SubmitFileName = CreateSubmitfileName(SubmitObjectID,
216 2                                                       SubmitElemID,
217 2
                                                          &CreateStatus)))
218 3     {
219 3       return -1;
220 2     }
221 2 #ifdef TESTMAIN

223 2     testmain_filename_G = strdup(SubmitFileName);

225 2 #else

227 2     if(0 != SetSESubmitFile(SubmitObjectID,
228 2                             SubmitElemID,
229 2                             SubmitFileName,
230 2                             &SetStatus))
231 3     {
232 3       return -1;
```

```
233  2      }
235  2  #endif
237  2      oflag = O_WRONLY|O_TRUNC|O_CREAT;
238  2      mode = S_IWRITE|S_IREAD;
239  1    }
240  1    else /* Read means the file exists, Lets get the name */
241  2    {
242  2      int GetStatus;
243  2  #ifdef TESTMAIN
245  2      strcpy(SubmitFileBuf, testmain_filename_G);
247  2  #else
249  2      if(0 != GetSESubmitFile(SubmitObjectID,
250  2                              SubmitElemID,
251  2                              SubmitFileBuf, E_PATH_MAX,
252  2                              &GetStatus))
253  3      {
254  3        return -1;
255  2      }
257  2  #endif
259  2      SubmitFileName = SubmitFileBuf;
260  2      oflag = O_RDONLY;
261  2      mode = S_IREAD;
262  1    }
264  1    do
265  2    {
267  2          fd = open(SubmitFileName, oflag, mode); /* No Body */
268  1    }
269  1    while((-1 == fd) && (EINTR == errno));
273  1    if(fd < 0)
274  2    {
275  2      *status = errno;
276  2      return -1;
277  1    }
279  1    /* Free is if was malloc'ed */
280  1    if((NULL != SubmitFileName) &&
281  1       (SubmitFileBuf != SubmitFileName))
282  2    {
283  2      free(SubmitFileName);
284  1    }
285  1    return fd;
286    }
```



```
289  /*
290   * CloseSubmitFile()
291   *  To be called to close a submit file. The submitfile
292   *  will be simply closed if the file was read only (See
293   *  WriteAcces arg). If the close is for a file opened for
294   *  write then a trailer will be appended to the submit
295   *  file and the mod will be set to Owner read permission
296   *  only and the file will be closed.
297   *
298   * Args:
299   *  (I) int submit_fd -- the submit file to close.
300   *  (I) boolean_ty WriteAccess -- TRUE if opening for write.
                                                    FALSE for read.
301   *  (O) int *status -- errno that caused failure.
302   *
303   * Returns: int
304   *  0 for success
305   *  non-zero for failue, check status.
306   *
307   *
308   */
310  int CloseSubmitFile(int submit_fd,
311                      boolean_ty WriteAccess,
312                      int *status)
314  1  {
315  1      int temp_status;
316  1      if(TRUE == WriteAccess)
317  2      {
318  2        /* If this close is for a write access,
319  2         * then write the trailor to the submit file
320  2         * And tighten up the permissions.
321  2         */
323  2        if(0 != WriteTrailorToSubmitFile(submit_fd, status))
324  3        {
325  3           return -1;
326  2        }
328  2        while(-1 == (temp_status = fchmod(submit_fd, O_RDONLY)) &&
329  2               EINTR == errno)
330  2           ; /* no body */
332  2        if(-1 == temp_status)
333  3        {
334  3           *status = errno;
335  3           return -1;
336  2        }
338  1      }
340  1      while((-1 == (temp_status = close(submit_fd))) &&
341  1             (EINTR == errno))
342  1        ; /* no body */
344  1      if(-1 == temp_status)
345  2      {
346  2        *status = errno;
347  2        return -1;
348  1      }
350  1      *status = 0;
351  1      return 0;
```

```
352 }
```



```
355 /*
356  * Internal use only.
357  */
358 static int
359 WriteTrailorToSubmitFile(const int submit_fd,
360                          int *status)
361 1 {
362 1   if(1 != Write(submit_fd, "!", 1))
363 2   {
364 2     *status = errno;
365 2     return -1;
366 1   }
367 1   return 0;
368 }
```

```
370  /*
371   * WriteBitfileInfoToSubmitFile()
372   *
373   * Args:
374   *  (I) int submit_fd -- Opened for write submit file descriptor.
375   *  (I) ebfs_uid_ty *ebfs_dirp -- Present bitfile parent bitfile
                                                                  directory.
376   *  (I) ebfs_uid_ty *ebfs_filep -- Present bitfile.
377   *  (I) ushort ce_mode --  mode of file from catalogs.
378   *  (I) int file_renamed_size --  Does the bitfile have a filename
                                                                  change.
379   *  (I) char *file_renamed --  Renamed file name to substiute in xcpio
                                                                  stream.
380   *  (I) int directive_size --  Does the bitfile an directive --
                                                              extended attribute.
381   *  (I) char *directive  --  Renamed file name to substiute in xcpio
                                                                  stream.
382   *  (I) u_hyper filesize -- File size.
383   *  (I/O) ebfs_uid_ty *prev_ebfs_dirp -- memset to zero for the first
                                                                     by
384   *             caller,
                       Must be maintained while writing to this submit file.
385   *
386   * if file_renamed_size == 0  then file_renamed will be NULL this
                                                                  indicates
387   * there was no rename.
388   *
389   * Returns:
390   *  0 for success.
391   *  non-zero for failure.
392   */
393  int
394  WriteBitfileInfoToSubmitFile(const int submit_fd,
395                               const ebfs_uid_ty *ebfs_dirp,
396                               const ebfs_uid_ty *ebfs_filep,
397                               const unsigned short ce_mode,
398                               const int file_renamed_size,
                                                  /*Needed for Netware */
399                               const char *file_renamed,
                                                   /*NULL if no rename*/
400                               const int directive_size,
401                               const char *directives,
402                               const u_hyper filesize,
403                               ebfs_uid_ty *prev_ebfs_dirp)
404  1 {
405  1     char             ebfsdirstr[34];
406  1     char             ebfsbfstr[34];
407  1     int              ret_Write = 0;
408  1     char             buf[100];
409  1     ebfs_uid_ty ZERO_ebfsp;
410  1     int              nbytes;

412  1     memset(buf, 0, 100);
413  1     memset(&ZERO_ebfsp, 0, sizeof(ZERO_ebfsp));
414  1     memset(ebfsdirstr, 0, 34);
415  1     memset(ebfsbfstr, 0, 34);


418  1     if((NULL == ebfs_dirp) && (NULL == ebfs_filep) && (
                                                 NULL == prev_ebfs_dirp))
419  2     {
420  2       return -1;
421  1     }

423  1     if ((0 == (memcmp(&ZERO_ebfsp, ebfs_dirp, sizeof(ebfs_dirp)))) &&
```



```
424  1          (0 == (memcmp(&ZERO_ebfsp, ebfs_filep, sizeof(ebfs_filep))))
425  2     {
426  2       return -1;
427  1     }


430  1     memcpy(prev_ebfs_dirp, ebfs_dirp, sizeof(prev_ebfs_dirp));
431  1     ebfsid2str_lz(ebfs_dirp, ebfsdirstr);

433  1     ebfsid2str_lz(ebfs_filep, ebfsbfstr);

435  1     (void)sprintf(buf, "%s,%s,%c", ebfsdirstr, ebfsbfstr,
436  1                   (S_ISDIR(ce_mode)) ? 'd' : '0');

438  1     nbytes = strlen(buf);
439  1     ret_Write = Write(submit_fd, buf, nbytes);
440  1     if(ret_Write != nbytes)
441  2     {
442  2       return -1;
443  1     }

445  1     memset(buf, 0, 100);

447  1     if((NULL != file_renamed) && (file_renamed_size > 0))
448  2     {
449  2          (void)sprintf(buf, "n%04x", file_renamed_size);
450  2          ret_Write = Write(submit_fd, buf, strlen(buf));
451  2          if(ret_Write != strlen(buf))
452  3          {
453  3            return -1;
454  2          }

456  2          ret_Write = Write(submit_fd, file_renamed, file_renamed_size);

457  2          if(ret_Write != file_renamed_size)
458  3          {
459  3            return -1;
460  2          }
461  1     }

463  1    if((NULL != directives) && (directive_size > 0))
464  2     {
465  2          (void)sprintf(buf, "D%04x", directive_size);
466  2          ret_Write = Write(submit_fd, buf, strlen(buf));
467  2          if(ret_Write != strlen(buf))
468  3          {
469  3            return -1;
470  2          }

472  2          ret_Write = Write(submit_fd, directives, directive_size);
473  2          if(ret_Write != directive_size)
474  3          {
475  3            return -1;
476  2          }

478  1     }

480  1     if (hyper_is_ulong(filesize))
481  2     {
482  2          sprintf(buf, ",%08x\n", filesize.low);
483  1     }
484  1     else
485  2     {
486  2          sprintf(buf, ",%08x%08x\n", filesize.high, filesize.low);
487  1     }
```

```
489 1     ret_Write = Write(submit_fd, buf, strlen(buf));
490 1     if(ret_Write != strlen(buf))
491 2     {
492 2       return -1;
493 1     }

495 1     return 0;
496   }
```



```
499   static ssize_t
500   Write(int fildes, const void *buf, size_t nbyte)
501 1 {
502 1   int ret_write;
503 1   while(-1 == (ret_write = (write(fildes, buf, nbyte))) &&
504 1           (EINTR == errno || EAGAIN == errno))
505 1     ;
506 1   return ret_write;
507   }
```

```
509
510  static ssize_t
511 1 Read(int fildes, void *buf, size_t nbyte)
512 1 {
513 1   int ret_read;
514 1   while(-1 == (ret_read = (read(fildes, buf, nbyte))) &&
515 1           (EINTR == errno || EAGAIN == errno))
516 1     ;
516 1   return ret_read;
517  }
```



```
519  /*
520   * ReadBitfileInfoFromSubmitFile()
521   *  Reads from the submit_fd one bitfiles record.
                                        If the ! is encounter where a
522   *  bitfile record should be it returns 0.
                                    If a 1 is returned one bitfile record
523   *  was read from the submit_fd.
                            If a -1 is return then the submit file format is
524   *  corrupt. This error is FATAL.
525   *
526   * NOTES: prev_ebfs_dirp should be maintained and memset to zero prior
                                                            to the first
527   *   call to ReadBitfileInfoFromSubmitFile().
528   *
529   * Side Effects:
530   *  Reads from submit_fd.
531   *
532   * Args:
533   *  (I) int submit_fd -- Opened for write submit file descriptor.
534   *  (O) ebfs_uid_ty *ebfs_dirp -- Present bitfile parent bitfile
                                                              directory.
535   *  (O) ebfs_uid_ty *ebfs_filep -- Present bitfile.
536   *  (O) boolean_ty *is_bfile_dir --  Does bitfile represent directory (
                                                              S_IFDIR)
537   *  (O) int file_renamed_size --  Does the bitfile have a filename
                                                                change.
538   *  (O) char **file_renamed --  Renamed file name to substiute in
                                                            xcpio stream.
539   *  (O) int *directive_size --  Does the bitfile an directive --
                                                          extended attribute.
540   *  (O) char **directive  --  Renamed file name to substiute in xcpio
                                                                stream.
541   *  (O) u_hyper *filesize -- File size.
542   *  (I/O) ebfs_uid_ty *prev_ebfs_dirp -- memset to zero for the first
                                                                    by
543   *             caller,
                      Must be maintained while writing to this submit file.
544   *
545   *
546   * Returns:
547   *  int 0 for end of file encountered.
548   *  int 1 for one bitfile info record read.
549   *  -1 for failure.
550   */
551  int
552  ReadBitfileInfoFromSubmitFile(const int submit_fd,
553                                ebfs_uid_ty *ebfs_dirp,
554                                ebfs_uid_ty *ebfs_filep,
555                                boolean_ty *is_bfile_dir,
556                                int *renamed_file_size,
557                                char **file_renamed,
558                                u_hyper *filesize,
559                                int *directives_size,
560                                char **directives,
561                                ebfs_uid_ty *prev_ebfs_dirp)

563 1 {
564 1   char temp_read_buffer[5000]; /* 4KB + 4 */
565 1   char *temp_read_buf_ptr = temp_read_buffer;
566 1   char temp_ebfs_str_buffer[34];
567 1   char *temp_ebfs_str_ptr = temp_ebfs_str_buffer;
568 1   char temp_attr_size_buffer[8];
569 1   char *temp_attr_size_ptr = temp_attr_size_buffer;
570 1   char temp_file_size_buffer[17];
571 1   char *temp_file_size_ptr = temp_file_size_buffer;
```

```
572 1   char temp_look;
573 1   int ret_Read;
574 1   int index;

576 1   boolean_ty have_read_filesize = FALSE;
577 1   boolean_ty have_not_started_filesize = FALSE;
578 1   boolean_ty directive_read = FALSE;
579 1   boolean_ty renamed_read = FALSE;

581 1   memset(temp_read_buf_ptr, 0, 5000);
582 1   memset(temp_ebfs_str_ptr, 0, 34);

584 1   ret_Read = Read(submit_fd, &temp_look, 1);
585 1   if (1 != ret_Read)
586 2   {
587 2     return -1;
588 1   }
589 1   if ('!' == temp_look)
590 2   {
591 2     /* This is an end of file condition */
592 2     return 0;
593 1   }
594 1   else if (',' == temp_look)
595 2   {
596 2     memcpy(ebfs_dirp, prev_ebfs_dirp, sizeof(ebfs_uid_ty));
597 1   }
598 1   else if(isxdigit(temp_look))
599 2   {

601 2     temp_ebfs_str_ptr[0] = temp_look;
602 2     temp_ebfs_str_ptr++;
603 2     /* Now read the next 31 plus the comma */
604 2     ret_Read = Read(submit_fd, temp_read_buf_ptr, 32);

606 2     if ((32 != ret_Read) || (',' != temp_read_buf_ptr[31]))
607 3     {
608 3       return -1;
609 2     }
610 2     for(index = 0; index < 31; index++, temp_ebfs_str_ptr++)
611 3     {
612 3       if(isxdigit(temp_read_buf_ptr[index]))
613 4       {
614 4          *temp_ebfs_str_ptr = temp_read_buf_ptr[index];
615 3       }
616 3       else
617 4       {
618 4          return -1;
619 3       }
620 2     }
621 2     temp_ebfs_str_ptr = temp_ebfs_str_buffer;

623 2     str_lz2ebfsid(temp_ebfs_str_ptr, ebfs_dirp);
624 2     memcpy(prev_ebfs_dirp, ebfs_dirp, sizeof(ebfs_uid_ty));
625 1   }
626 1   else
627 2   {
628 2     return -1;
629 1   }

631 1   temp_read_buf_ptr = temp_read_buffer;
632 1   temp_ebfs_str_ptr = temp_ebfs_str_buffer;
633 1   memset(temp_read_buf_ptr, 0, 5000);
634 1   memset(temp_ebfs_str_ptr, 0, 34);

636 1   /* the bitfile id[32] + the comma + file type char */
637 1   ret_Read = Read(submit_fd, temp_read_buf_ptr, 34);
```



```
638 1   if ((34 != ret_Read) || (',' != temp_read_buf_ptr[32]))
639 2   {
640 2     return -1;
641 1   }
642 1   for(index = 0; index < 32; index++, temp_ebfs_str_ptr++)
643 2   {
644 2     if(isxdigit(temp_read_buf_ptr[index]))
645 3     {
646 3       *temp_ebfs_str_ptr = temp_read_buf_ptr[index];
647 2     }
648 2     else
649 3     {
650 3       return -1;
651 2     }
652 1   }
653 1   temp_ebfs_str_ptr = temp_ebfs_str_buffer;

655 1   str_lz2ebfsid(temp_ebfs_str_ptr, ebfs_filep);
656 1   if(temp_read_buf_ptr[33] == 'd')
657 2   {
658 2     *is_bfile_dir = TRUE;
659 1   }
660 1   else if (temp_read_buf_ptr[33] == '0')
661 2   {
662 2     *is_bfile_dir = FALSE;
663 1   }
664 1   else
665 2   {
666 2     return -1;
667 1   }

669 1   temp_read_buf_ptr = temp_read_buffer;
670 1   temp_ebfs_str_ptr = temp_ebfs_str_buffer;

672 1   memset(temp_ebfs_str_ptr, 0, 34);
673 1   memset(temp_attr_size_ptr, 0, 5);
674 1   memset(temp_file_size_ptr, 0, 17);

676 1   /* We read 5 bytes here because it will be either
677 1    * 1) renamed file indicater and length of the name if renamed file.
678 1    * 2) extended directive indicater and lenght of the directive.
679 1    * 3) the first 5 charactors of the file size.
680 1    */

682 1   while(!have_read_filesize)
683 2   {
684 2     memset(temp_read_buf_ptr, 0, 5000);

686 2     ret_Read = Read(submit_fd, temp_read_buf_ptr, 5);

688 2     if (5 != ret_Read)
689 3     {
690 3       return -1;
691 2     }
692 2     if(('n' == temp_read_buf_ptr[0]) || ('D' == temp_read_buf_ptr[0]))
693 3     {
694 3       unsigned long attr_size = 0;
695 3       char *attr_temp;


698 3       strncpy(temp_attr_size_ptr, temp_read_buf_ptr + 1, 4);
699 3       attr_size =  strtoul(temp_attr_size_ptr, (char **) NULL, 16);

701 3       attr_temp = (char *) calloc(1, (attr_size + 1));
702 3       if(NULL == attr_temp)
703 4       {
```

```
704 4          return -1;
705 3       }
706 3       ret_Read = Read(submit_fd, attr_temp, attr_size);
707 3       if (attr_size != ret_Read)
708 4       {
709 4          return -1;
710 3       }

712 3       if('n' == temp_read_buf_ptr[0])
713 4       {
714 4          renamed_read = TRUE;
715 4          *renamed_file_size = attr_size;
716 4          *file_renamed = attr_temp;
717 4          continue;
718 3       }
719 3       if('D' == temp_read_buf_ptr[0])
720 4       {
721 4          directive_read = TRUE;
722 4          *directives_size = attr_size;
723 4          *directives = attr_temp;
724 4          continue;

726 3       }
727 2     }
728 2     else if (',' == temp_read_buf_ptr[0])
729 3     {
730 3       memcpy(temp_file_size_ptr, &temp_read_buf_ptr[1], 4);
731 3       temp_file_size_ptr += 4;
732 3       /*
733 3        * If the file size is a ulong then the 5 byte should be '\n'.
734 3        */
735 3       ret_Read = Read(submit_fd, temp_file_size_ptr, 5);
736 3       if (5 != ret_Read)
737 4       {
738 4          return -1;
739 3       }
740 3       if(temp_file_size_ptr[4] == '\n')
741 4       {
742 4          temp_file_size_ptr[4] = '\0';
743 4          temp_file_size_ptr = temp_file_size_buffer;

745 4          for(index = 0; index < 8; index++)
746 5          {
747 5            if(!isxdigit(temp_file_size_ptr[index]))
748 6            {
749 6              return -1;
750 5            }
751 4          }
752 4          *filesize =  ul_to_uh(strtoul(temp_file_size_ptr, (
                                                  char **) NULL, 16));
753 4          have_read_filesize = TRUE;
754 3       }
755 3       else if (isxdigit(temp_file_size_ptr[4]))
756 4       {
757 4          temp_file_size_ptr = &temp_file_size_buffer[9]; /* alton */
758 4          ret_Read = Read(submit_fd, temp_file_size_ptr, 8);
759 4          if ((8 != ret_Read) || ('\n' != temp_file_size_ptr[7]))
760 5          {
761 5            return -1;
762 4          }

764 4          temp_file_size_ptr = temp_file_size_buffer;
765 4          temp_file_size_ptr[16] = '\0';
766 4          for(index = 0; index < 16; index++)
767 5          {
768 5            if(!isxdigit(temp_file_size_ptr[index]))
```



```
769 6            {
770 6              return -1;
771 5            }
772 4          }
773 4          /* hack lets use a large enough buffer
774 4             to put a "0x" at the beginning of our hexidecimal uhyper
                                                                  sting */


777 4          memset(temp_ebfs_str_buffer,0, 34);
778 4          temp_ebfs_str_buffer[0] = '0';
779 4          temp_ebfs_str_buffer[1] = 'x';
780 4          temp_ebfs_str_buffer[2] = '\0';
781 4          strcat(&temp_ebfs_str_buffer[2], temp_file_size_ptr);

783 4          if(0!= string_to_u_hyper(temp_ebfs_str_buffer, filesize))
784 5          {
785 5            return -1;
786 4          }

788 4          have_read_filesize = TRUE;
789 3       }
790 3       else
791 4       {
792 4          return -1;
793 3       }
794 2     }
795 2     else
796 3     {
797 3       return -1;
798 2     }
799 1   } /* end while () */
800 1   return 0;
801   } /* end of ReadBitfileInfoFromSubmitFile() */
```

```
803     /*
804      * Convert an string to EBFS ID.
805      */
806     int
807     str_lz2ebfsid(char *ebfsdirstr, ebfs_uid_ty *ebfs_p)
808   1 {
809   1   int index;
810   1   unsigned int *upptr = (unsigned int *)ebfs_p;

812   1   memset(ebfs_p, 0, 32);

814   1   for(index=0; index < 32; index++)
815   2   {
816   2     if(!isxdigit(ebfsdirstr[index]))
817   3     {
818   3       return -1;
819   2     }
820   2     else
821   3     {
822   3       if((index != 0) && (0 == (index % 8)))
823   4       {
824   4           upptr++;
825   3       }
826   3       *upptr = *upptr << 4;
827   3       *upptr = *upptr + hdigit_conv[ebfsdirstr[index]];
828   2     }
829   1   }
830   1   return 0;
831     } /* end of str_lz2ebfsid() */
```



```
833     /*
834      * Convert an EBFS ID to string form.  Elide leading zeros.
835      */

837     static void
838     ebfsid2str_lz(const ebfs_uid_ty *ebfsidp,
839                   register char *buf)
840     #if 0
841   1 {
842   1     register char *p;
843   1     register char *q;
844   1     unsigned long longvals[4];
845   1     int i;
846   1     char tmpbuf[ 33 ];
847   1     char *hexdigits = "0123456789abcdef";


850   1     memcpy(longvals, ebfsidp, 16);

852   1     q = tmpbuf;
853   1     q = tmpbuf;

855   1     for (i = 0; i < 4; i++)
856   2     {
857   2         int j;
858   2         register unsigned long ul;

860   2         q += 8;
861   2         ul = longvals[i];

863   2         for (j = 0; j < 8; j++)
864   3         {
865   3             *--q = hexdigits[ LONG2INT(ul & 0xF) ];
866   3             ul >>= 4;
867   2         }
868   2         q += 8;
869   1     }

871   1     tmpbuf[32] = '\0';

873   1     /*
874   1      * Skip over leading 0 characters
875   1      */
876   1     for (q = tmpbuf; *q == '0'; q++)
877   1       ;
878   1     /*
879   1      * Copy the rest, up to and including the comma, to output buf
880   1      */

882   1     p = buf;
883   1     p = buf;
884   1     while ((*p++ = *q++) != '\0')
885   2     {
886   2         /* null */
887   1     }
888     }       /* end of ebfsid2str_lz() */
```

```
889     #endif
890   1 {
891   1   int index;
892   1   unsigned int *upptr = (unsigned int *)ebfsidp;
893   1   char *buf_ptr = buf;
894   1   char *hexdigits = "0123456789abcdef";

896   1   for(index=0; 4 > index ; index++, upptr++)
897   2   {
898   2     buf_ptr[0] =     hexdigits[ ((*upptr & 0xF0000000) >> 28) ];
899   2     buf_ptr[1] =     hexdigits[ ((*upptr & 0x0F000000) >> 24) ];
900   2     buf_ptr[2] =     hexdigits[ ((*upptr & 0x00F00000) >> 20) ];
901   2     buf_ptr[3] =     hexdigits[ ((*upptr & 0x000F0000) >> 16) ];
902   2     buf_ptr[4] =     hexdigits[ ((*upptr & 0x0000F000) >> 12) ];
903   2     buf_ptr[5] =     hexdigits[ ((*upptr & 0x00000F00) >> 8) ];
904   2     buf_ptr[6] =     hexdigits[ ((*upptr & 0x000000F0) >> 4) ];
905   2     buf_ptr[7] =     hexdigits[ ((*upptr & 0x0000000F) >> 0) ];

907   2     buf_ptr += 8;
908   1   }
909   1   return;
910     } /* end of ebfsid2str_lz() */
```



```
916   /********** Bitfile record format ***************
917   D32,B32,T,nllllrrrrrrrrrXllllxxxxxxxxxxxx,fsfsfsfs[fsfsfsfs]\n

919   D32 -- is the bitfile directory id in 32 hexdecimal ascii chars.
920   If not present the previous bitfile directory will be assumed.

922   D32 -- is the bitfile id in 32 hexdecimal ascii chars.

924   T -- Chararactor to indicate whether the file being restore
925   was a directory or not. This can be 'd' for directory file
926   type or '0' for a non-directory file type.

928   Optional nllllrrrrrrrrr
929   ----------------------
930   n -- Rename attribute indicator. This is an ascii 'n'.

932   llll -- length of the renamed file name.

934   rrrrrrrrr -- the renamed file name.

936   Optional Xllllxxxxxxxxxxx
937   ---------------------------

939   X -- Extended attributes directive indicator. This is an ascii 'X'.

941   llll -- length of the Extended attributes directive.

943   xxxxxxxxxxx -- Extended attribute that needs to be inserted into the
944   xcpio stream.

946   fsfsfsfs[fsfsfsfs] -- The file size in 8 or 16 hexidecimal ascii,
947   ulong or uhyper.

949   \n -- New line bitfile record delimiter.

951   ***************** End format section  **************************/


954   #ifdef TESTMAIN

956   int
957   bitfile_info_cmp(struct submitfile_bitfile_info *binfo1,
958                    struct submitfile_bitfile_info *binfo2)
959   1 {
960   1   int temp_cmp;

962   1   if(0 != (temp_cmp = strcmp(
                              binfo1 -> ebfs_dirp, binfo2 -> ebfs_dirp)))
963   2   {
964   2     return temp_cmp;
965   1   }
966   1   if(0 != (temp_cmp = strcmp(
                            binfo1 -> ebfs_filep, binfo2 -> ebfs_filep)))
967   2   {
968   2     return temp_cmp;
969   1   }

971   1   if(binfo1 -> is_bfile_dir != binfo2 -> is_bfile_dir)
972   1     return binfo1 -> is_bfile_dir - binfo2 -> is_bfile_dir;

974   1   if(binfo1 -> file_renamed_size != binfo2 -> file_renamed_size)
```

```
975 1     return binfo1 -> file_renamed_size - binfo2 -> file_renamed_size;

977 1   if((0 != binfo1 -> file_renamed_size) &&
978 1      (0 != (temp_cmp = strcmp(
                      binfo1 -> file_renamed, binfo2 -> file_renamed))))
979 2   {
980 2     return temp_cmp;
981 1   }

983 1   if(binfo1 -> directive_size != binfo2 -> directive_size)
984 1     return binfo1 -> directive_size - binfo2 -> directive_size;

986 1   if((0 != binfo1 -> directive_size) &&
987 1   (0 != (temp_cmp = strcmp(
                          binfo1 -> directives, binfo2 -> directives))))
988 2   {
989 2     return temp_cmp;
990 1   }

992 1   if(binfo1 -> filesize_high != binfo2 -> filesize_high)
993 1     return binfo1 -> filesize_high - binfo2 -> filesize_high;

995 1   if(binfo1 -> filesize_low != binfo2 -> filesize_low)
996 1     return binfo1 -> filesize_low - binfo2 -> filesize_low;

998 1   return 0;
999   }
```



```
1001  #define COPY_BINFO(testbuf, index, dirid, bitfileid, isdir, \
1002                     rename_sz, rename, direct_sz, direct, \
1003                     file_sz_high, file_sz_low) \
1004  \
1005  strcpy(testbuf[index].ebfs_dirp, dirid); \
1006  strcpy(testbuf[index].ebfs_filep, bitfileid); \
1007  testbuf[index].is_bfile_dir = isdir; \
1008  testbuf[index].file_renamed_size = rename_sz; \
1009  if (rename)  testbuf[index].file_renamed = strdup(rename); \
1010  else testbuf[index].file_renamed = NULL; \
1011  testbuf[index].directive_size = direct_sz; \
1012  if (direct)  testbuf[index].directives = strdup(direct); \
1013  else testbuf[index].directives = NULL; \
1014  testbuf[index].filesize_high = file_sz_high; \
1015  testbuf[index].filesize_low = file_sz_low;

1017  struct submitfile_bitfile_info testbuffer[12];
1018  void initbuffer(struct submitfile_bitfile_info *testbuffer)
1019 1 {



1022 1     /* 0 */
1023 1   COPY_BINFO(testbuffer, 0,
1024 1              "ddddddddddddddddddddddddddddddddd",
1025 1              "bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb",
1026 1              TRUE, 0, NULL, 0, NULL, 0, 0);

1028 1   /* 1 */
1029 1   COPY_BINFO(testbuffer, 1,
1030 1              "12345678901234567890123456789012",
1031 1              "0123456789abcdef0123456789abcdef",
1032 1              FALSE, 0, NULL, 0, NULL, 0, 5);

1034 1   /* 2 */
1035 1   COPY_BINFO(testbuffer, 2,
1036 1              "67890123678901236789012367890123",
1037 1              "0000456789abcdef0123456789abcdef",
1038 1              FALSE, 10, "/one/two/x", 0, NULL, 0, 5);

1040 1   /* 3 */
1041 1   COPY_BINFO(testbuffer, 3,
1042 1              "67890123678901236789012367890123",
1043 1              "0000456789abcdef0123456789abcdef",
1044 1              FALSE, 10, "/one/two/x", 11, "DIRECT=one", 2, 5);

1046 1   /* 4 */
1047 1   COPY_BINFO(testbuffer, 4,
1048 1              "6789dfec26789dfec6789dfec6789df7",
1049 1              "badbadbadbadbadbadbadbadbadbadba",
1050 1              FALSE, 10, "/one/two/x", 9, "DIRECT=x", 2, 5);

1052 1   /* 5 */
1053 1   COPY_BINFO(testbuffer, 5,
1054 1              "feedfeedfeedfeedfe89012367890123",
1055 1              "feedfeedfeedfeedfeedfeedfeedfeed",
1056 1              FALSE, 10, "/one/two/x", 0, NULL, 2, 5);

1058 1   /* 6 */
1059 1   COPY_BINFO(testbuffer, 6,
1060 1              "67890123678901236789012367890123",
1061 1              "0000456789abcdef0123456789abcdef",
1062 1              FALSE, 0, NULL, 11, "DIRECT=one", 2, 5);

1064 1   /* 7 */
1065 1   COPY_BINFO(testbuffer, 7,
```

```
1066 1              "67890123678901236789012367890123",
1067 1              "0000456789abcdef0123456789abcdef",
1068 1              TRUE, 10, "/dir1/dir2", 0, NULL, 2, 5);

1070 1   /* 8 */
1071 1   COPY_BINFO(testbuffer, 8,
1072 1              "67890123678901236789012367890123",
1073 1              "0000456789abcdef0123456789abcdef",
1074 1              FALSE, 10, "/one/two/x", 11, "DIRECT=one", 2, 5);

1076 1   /* 9 */
1077 1   COPY_BINFO(testbuffer, 9,
1078 1              "67890123678901236789012367890123",
1079 1              "0000456789abcdef0123456789abcdef",
1080 1              FALSE, 10, "/one/two/x", 11, "DIRECT=one", 2, 5);

1082 1   /* 10 */
1083 1   COPY_BINFO(testbuffer, 10,
1084 1              "67890123678901236789012367890123",
1085 1              "0000456789abcdef0123456789abcdef",
1086 1              FALSE, 10, "/one/two/x", 11, "DIRECT=one", 2, 5);

1088 1   /* 11 */
1089 1   COPY_BINFO(testbuffer, 11,
1090 1              "67890123678901236789012367890123",
1091 1              "0000456789abcdef0123456789abcdef",
1092 1              FALSE, 10, "/one/two/x", 11, "DIRECT=one", 2, 5);

1094    }
```



```
1096    int main()
1097 1  {
1098 1    ebfs_uid_ty bitfileID;
1099 1    ebfs_uid_ty dirID;
1100 1    ebfs_uid_ty PrevdirID;
1101 1    u_hyper filesize;
1102 1    int status;
1103 1    int fd;
1104 1    int index;
1105 1    struct submitfile_bitfile_info readbuffer[12];

1107 1    memset(&PrevdirID, 0, sizeof(ebfs_uid_ty));

1109 1    memset(&readbuffer, 0,
1110 1           12 * sizeof(struct submitfile_bitfile_info));

1112 1    initbuffer(testbuffer);

1114 1    fd = OpenSubmitFile(TRUE, 1, 1, &status);

1116 1    for(index = 0; index < 12; index++)
1117 2    {


1120 2      (void)str_lz2ebfsid(testbuffer[index].ebfs_dirp, &dirID);
1121 2      (void)str_lz2ebfsid(testbuffer[index].ebfs_filep, &bitfileID);
1122 2      filesize.low = testbuffer[index].filesize_low;
1123 2      filesize.high = testbuffer[index].filesize_high;

1125 2      if(0 != WriteBitfileInfoToSubmitFile(fd,
1126 2                                           &dirID,
1127 2                                           &bitfileID,
1128 2                                           testbuffer[index].
                                                          is_bfile_dir,
1129 2                                           testbuffer[index].
                                                    file_renamed_size,
1130 2                                           testbuffer[index].
                                                       file_renamed,
1131 2                                           testbuffer[index].
                                                         directive_size,
1132 2                                           testbuffer[index].directives,

1133 2                                           filesize,
1134 2                                           &PrevdirID))
1135 3      {
1136 3        /* error occurred */;
1137 2      }
1138 1    }
1139 1    (void)CloseSubmitFile(fd,
1140 1                          TRUE,
1141 1                          &status);

1143 1    fd = OpenSubmitFile(FALSE, 1, 1, &status);


1146 1    memset(&PrevdirID, 0, sizeof(ebfs_uid_ty));

1148 1    for(index = 0; index < 12; index++)
1149 2    {
1150 2      ReadBitfileInfoFromSubmitFile(fd,
1151 2                                    &dirID,
1152 2                                    &bitfileID,
1153 2                                    &(readbuffer[index].is_bfile_dir),
1154 2                                    &(
                                      readbuffer[index].file_renamed_size),
```

```
1155  2                                   &(readbuffer[index].file_renamed),
1156  2                                   &filesize,
1157  2                                   &(
                                           readbuffer[index].directive_size),
1158  2                                   &(readbuffer[index].directives),
1159  2                                   &PrevdirID);

1161  2      ebfsid2str_lz(&dirID, readbuffer[index].ebfs_dirp);
1162  2      ebfsid2str_lz(&bitfileID, readbuffer[index].ebfs_filep);

1164  2      readbuffer[index].filesize_low = filesize.low;
1165  2      readbuffer[index].filesize_high = filesize.high;

1167  2      if(0 == bitfile_info_cmp(&readbuffer[index],
1168  2                               &testbuffer[index]))
1169  2        fprintf(stdout, "\nThe bitfile info matched %d\n", index);
1170  2      else
1171  2        fprintf(
              stdout, "\nThe bitfile info DID NOT matched %d\n", index);
1172  1    }

1174  1    (void)CloseSubmitFile(fd,
1175  1                          FALSE,
1176  1                          &status);

1178  1    unlink((const char *)testmain_filename_G);
1179  1    free(testmain_filename_G);
1180  1    testmain_filename_G = NULL;

1182  1    return 0;
1183      }
```



```
1185  #endif
```

```
1   /*
2   ** Copyright 1996,1997 EMC Corporation
3   */


6   /* EDMRESubmitApi.cc
7    *
8    *
9    * Mission Statement:  file that contains an API to manage the Submit
                                                                    objects.
10   *
11   * Primary Data Acted On:
12   *
13   * Compile-Time Options:
14   *
15   * Basic idea here:
16   *                     A few calls to manage the Submit objects.
                                                                    This is
17   *                     used by the Process Manager thread.
18  */

20  #if !defined(lint)
21  static char     RCS_id [] = "@(#)$RCSfile: EDMRESubmitApi.cc,v $ "
22                              "$Revision: 1.0 $ "
23                              "$Date: 1997/02/06 20:49:15 $" ;
24  #endif

26  #include <esl/c_portable.h>
27  #include <esl/ep_xopen.h>
28  #include <esl/inout.h>

30  #include <stdlib.h>
31  #include <sys/types.h>
32  #include <pthread.h>

34  #include <restore/EDMRESubmitApi.h>

36  static unsigned int  numberOfSubmitObjects = 0;
37  static RWBinaryTree submitObjects;
38  static pthread_mutex_t  G_submitmutex = PTHREAD_MUTEX_INITIALIZER;

40  /*************************************************************************
                                                                    ********
41  **
42  ** Routine:  lockSubmitMutex
43  **
44  ** Inputs:   None
45  **
46  ** Outputs:  None
47  **
48  ** Return Codes:
49  **           None
50  **
51  ** Purpose:  Lock the mutex for the submit object
52  **
53  *************************************************************************
                                                                    ********
54  */

56  static void
57  lockSubmitMutex()
58 1 {
59 1      static boolean_ty first = TRUE;

61 1      if (first == TRUE)
62 2      {
```



```
63 2            first = FALSE;
64 2            pthread_mutex_init(&G_submitmutex, NULL);
65 1        }

67 1        pthread_mutex_lock(&G_submitmutex);
68      }
```

```
/*****************************************************************************
                                                                        ********
**
** Routine:  unlockSubmitMutex
**
** Inputs:   None
**
** Outputs:  None
**
** Return Codes:
**           None
**
** Purpose:  Unlock the mutex for the submit object
**
*****************************************************************************
                                                                        ********
*/

static void
unlockSubmitMutex()
{
    pthread_mutex_unlock(&G_submitmutex);
}
```



```
/*****************************************************************************
                                                                        ********
**
** Routine:  LookupSubmitElement
**
** Inputs:   int ID - submit object ID associated with element
**           int elementID - submit element ID associated with element
**
** Outputs:  int *status - status of the function if an error occured
**           EDMRESubmitElement **se - place to put the pointer to the
                                                                        element
**
** Return Codes:
**           int - 0 for success or non-zero for failure
**
** Purpose:  Finds a submit element based on the submit ID and the
                                                                        element ID.
**
*****************************************************************************
                                                                        ********
*/
int
RemoveSubmitFiles ()
{
  // In the future we might want to pass in the submit object
  // in case they become persistent stores.

  EDMRESubmitObj *so;
  RWBinaryTreeIterator *submitObjectsTree;

  // create a binary tree iterator so that we can get each submit
                                                                        object
  // and therefore get each submit element and remove its submit file
  submitObjectsTree = new RWBinaryTreeIterator(submitObjects);
  // if it is null we have a calloc failure
  if (NULL == submitObjectsTree)
    {
      return (-1);
    }

  //set the iterator to start at the begining
  submitObjectsTree->reset();

  // while we have objects to work with we must keep getting the
  // next element.  This also gets the first object once a reset
  // is performed like above.  the () operator sets to the next
  // element in the tree
  while(NULL !=(*submitObjectsTree)() )
    {
      //get the next object out of the tree
      so = (EDMRESubmitObj *)submitObjectsTree->key();
      so->deleteSubmitFiles();
    }
  //delete the submitObjects Iterator
  delete submitObjectsTree;
  // return success
  return (0);
}
```

```
147   int
148   LookupSubmitElement(int ID, int elementID, EDMRESubmitElement **se,
149                       int *status)
150 1 {
151 1     EDMRESubmitObj *so;
152 1     EDMRESubmitObj *ret;

154 1     if (status == NULL)
155 2     {
156 2         return -1;
157 1     }

159 1     *status = 0;

161 1     if (se == NULL)
162 2     {
163 2         *status = SUBMIT_BAD_PARAM;
164 2         return -1;
165 1     }

167 1     so = new EDMRESubmitObj();

169 1     if (so == NULL)
170 2     {
171 2         *status = SUBMIT_NO_MEMORY;
172 2         return -1;
173 1     }

175 1     so -> setSubmitID(ID);

177 1     lockSubmitMutex();

179 1     ret = (EDMRESubmitObj *) submitObjects.find(so);

181 1     delete so;

183 1     if (ret == NULL)
184 2     {
185 2         *status = SUBMIT_LOOKUP_FAILED;
186 2         unlockSubmitMutex();
187 2         return -1;
188 1     }

190 1     *se = ret -> getSubmitElement(elementID);

192 1     unlockSubmitMutex();

194 1     if (*se == NULL)
195 2     {
196 2         *status = SUBMIT_LOOKUP_FAILED;
197 2         return -1;
198 1     }

200 1     return 0;
201   }
```



```
203   /*****************************************************************************
                                                                          ********
204   **
205   ** Routine:  LookupSubmitObject
206   **
207   ** Inputs:   int ID - submit object ID
208   **
209   ** Outputs:  int *status - status of the function if an error occured
210   **
211   ** Return Codes:
212   **           int - 0 for success or non-zero for failure
213   **
214   ** Purpose:  Finds a submit object based on the submit ID.
215   **
216   *****************************************************************************
                                                                          ********
217   */

219   int
220   LookupSubmitObject(int ID, EDMRESubmitObj **so, int *status)
221 1 {
222 1     EDMRESubmitObj *tmpso;
223 1     EDMRESubmitObj *ret;

225 1     if (status == NULL)
226 2     {
227 2         return -1;
228 1     }

230 1     if (so == NULL)
231 2     {
232 2         *status = SUBMIT_BAD_PARAM;
233 2         return -1;
234 1     }

236 1     tmpso = new EDMRESubmitObj();

238 1     if (tmpso == NULL)
239 2     {
240 2         *status = SUBMIT_NO_MEMORY;
241 2         return -1;
242 1     }

244 1     tmpso -> setSubmitID(ID);

246 1     lockSubmitMutex();

248 1     ret = (EDMRESubmitObj *) submitObjects.find(tmpso);

250 1     unlockSubmitMutex();

252 1     delete tmpso;

254 1     if (ret == NULL)
255 2     {
256 2         *status = SUBMIT_LOOKUP_FAILED;
257 2         return -1;
258 1     }

260 1     *so = ret;

262 1     return 0;
263   }
```

```
265   /*****************************************************************************
                                                                          ********
266   **
267   ** Routine:  NewSubmitObject
268   **
269   ** Inputs:   NONE
270   **
271   ** Outputs:  int *status - status of the function if an error occured
272   **
273   ** Return Codes:
274   **           int - ID of the new submit Object
275   **
276   ** Purpose:  Creates a new submit object and inserts it in the submit
                                                                        tree.
277   **
278   *****************************************************************************
                                                                          ********
279   */

281   int
282   NewSubmitObject(int *status)
283 1 {
284 1     EDMRESubmitObj *so;
285 1     EDMRESubmitObj *ret;

287 1     if (status == NULL)
288 2     {
289 2         return 0;
290 1     }

292 1     so = new EDMRESubmitObj();

294 1     if (so == NULL)
295 2     {
296 2         *status = SUBMIT_NO_MEMORY;
297 2         return 0;
298 1     }

300 1     so -> setSubmitID(++numberOfSubmitObjects);

302 1     lockSubmitMutex();

304 1     ret = (EDMRESubmitObj *) submitObjects.insert(so);

306 1     unlockSubmitMutex();

308 1     if (ret == NULL)
309 2     {
310 2         *status = SUBMIT_INSERT_FAILED;
311 2         numberOfSubmitObjects--;
312 2         delete so;
313 2         return 0;
314 1     }

316 1     return numberOfSubmitObjects;
317   }
```



```
319   /*****************************************************************************
                                                                          ********
320   **
321   ** Routine:  NewSubmitElement
322   **
323   ** Inputs:   int ID - submit ID associated with new element
324   **
325   ** Outputs:  int *status - status of the function if an error occured
326   **
327   ** Return Codes:
328   **           int - ID of the new submit element
329   **
330   ** Purpose:  Creates a new submit element and inserts it in the
                                                                element tree.
331   **
332   *****************************************************************************
                                                                          ********
333   */

335   int
336   NewSubmitElement(int ID, int *status)
337 1 {
338 1     EDMRESubmitObj *so;
339 1     EDMRESubmitObj *ret;
340 1     int elementID = 0;

342 1     if (status == NULL)
343 2     {
344 2         return 0;
345 1     }

347 1     so = new EDMRESubmitObj();

349 1     if (so == NULL)
350 2     {
351 2         *status = SUBMIT_NO_MEMORY;
352 2         return 0;
353 1     }

355 1     so -> setSubmitID(ID);

357 1     lockSubmitMutex();

359 1     ret = (EDMRESubmitObj *) submitObjects.find(so);

361 1     delete so;

363 1     if (ret == NULL)
364 2     {
365 2         *status = SUBMIT_LOOKUP_FAILED;
366 2         unlockSubmitMutex();
367 2         return 0;
368 1     }

370 1     elementID =  ret -> newSubmitElement();

372 1     unlockSubmitMutex();

374 1     return elementID;
375   }
```

```
377  /*************************************************************************
                                                                        ********
378  **
379  ** Routine:  GetSEClientUserName
380  **
381  ** Inputs:   int ID - submit ID associated with element
382  **           int elementID - submit element ID associated with element
383  **           int maxsize - maximum size to fill the buffer with
384  **
385  ** Outputs:  int *status - status of the function if an error occured
386  **           char *buff - place to put the user name
387  **
388  ** Return Codes:
389  **           int - 0 for success or non-zero for failure
390  **
391  ** Purpose:  Gets the user name of the given client.
392  **
393  *************************************************************************
                                                                        ********
394  */

396  int
397  GetSEClientUserName(int ID, int elementID, char *buff, int maxsize,
398                      int *status)
399 1 {
400 1     EDMRESubmitElement *se;
401 1     int ret;

403 1     if (status == NULL)
404 2     {
405 2         return -1;
406 1     }

408 1     if (buff == NULL)
409 2     {
410 2         *status = SUBMIT_BAD_PARAM;
411 2         return -1;
412 1     }

414 1     ret = LookupSubmitElement(ID, elementID, &se, status);

416 1     if (ret != 0)
417 2     {
418 2         return ret;
419 1     }

421 1     se -> getClientUserName(buff, maxsize);

423 1     return 0;
424  }
```



```
426  /*************************************************************************
                                                                        ********
427  **
428  ** Routine:  GetSEWorkItemName
429  **
430  ** Inputs:   int ID - submit ID associated with element
431  **           int elementID - submit element ID associated with element
432  **           int maxsize - maximum size to fill the buffer with
433  **
434  ** Outputs:  int *status - status of the function if an error occured
435  **           char *buff - place to put the work item
436  **
437  ** Return Codes:
438  **           int - 0 for success or non-zero for failure
439  **
440  ** Purpose:  Gets the workitem name of the given submit element.
441  **
442  *************************************************************************
                                                                        ********
443  */

445  int
446  GetSEWorkItemName(int ID, int elementID, char *buff, int maxsize,
447                    int *status)
448 1 {
449 1     EDMRESubmitElement *se;
450 1     int ret;

452 1     if (status == NULL)
453 2     {
454 2         return -1;
455 1     }

457 1     if (buff == NULL)
458 2     {
459 2         *status = SUBMIT_BAD_PARAM;
460 2         return -1;
461 1     }

463 1     ret = LookupSubmitElement(ID, elementID, &se, status);

465 1     if (ret != 0)
466 1         return ret;

468 1     se -> getWorkItem(buff, maxsize);

470 1     return 0;
471  }
```

```
473  /*************************************************************************
                                                                        ********
474  **
475  ** Routine:  GetSETemplateName
476  **
477  ** Inputs:   int ID - submit ID associated with element
478  **           int elementID - submit element ID associated with element
479  **           int maxsize - maximum size to fill the buffer with
480  **
481  ** Outputs:  int *status - status of the function if an error occured
482  **           char *buff - place to put the template
483  **
484  ** Return Codes:
485  **           int - 0 for success or non-zero for failure
486  **
487  ** Purpose:  Gets the template name of the given submit element.
488  **
489  *************************************************************************
                                                                        ********
490  */

492  int
493  GetSETemplateName(int ID, int elementID, char *buff, int maxsize,
494                    int *status)
495  1 {
496  1     EDMRESubmitElement *se;
497  1     int ret;

499  1     if (status == NULL)
500  2     {
501  2         return -1;
502  1     }

504  1     if (buff == NULL)
505  2     {
506  2         *status = SUBMIT_BAD_PARAM;
507  2         return -1;
508  1     }

510  1     ret = LookupSubmitElement(ID, elementID, &se, status);

512  1     if (ret != 0)
513  1         return ret;

515  1     se -> getTemplate(buff, maxsize);

517  1     return 0;
518  }
```



```
520  /*************************************************************************
                                                                        ********
521  **
522  ** Routine:  GetSEIsTrailSetAlternate
523  **
524  ** Inputs:   int ID - submit ID associated with element
525  **           int elementID - submit element ID associated with element
526  **
527  ** Outputs:  int *status - status of the function if an error occured
528  **
529  ** Return Codes:
530  **           boolean_ty - TRUE is yes and FALSE otherwise
531  **
532  ** Purpose:  Gets the Is-Trail-Set-Alternate boolean from submit
                                                                        element.
533  **
534  *************************************************************************
                                                                        ********
535  */

537  boolean_ty
538  GetSEIsTrailSetAlternate(int ID, int elementID,
539                           int *status)
540  1 {
541  1     EDMRESubmitElement *se;
542  1     int ret;

544  1     if (status == NULL)
545  2     {
546  2         return FALSE;
547  1     }

549  1     ret = LookupSubmitElement(ID, elementID, &se, status);

551  1     if (ret != 0)
552  1         return FALSE;

554  1     return se -> IsAlternateTrailset();
555  }
```

```
557  /*************************************************************************
                                                                        ********
558  **
559  ** Routine:  GetSESourceClientName
560  **
561  ** Inputs:   int ID - submit ID associated with element
562  **           int elementID - submit element ID associated with element
563  **           int maxsize - maximum size to fill the buffer with
564  **
565  ** Outputs:  int *status - status of the function if an error occured
566  **           char *buff - place to put the client name
567  **
568  ** Return Codes:
569  **           int - 0 for success or non-zero for failure
570  **
571  ** Purpose:  Gets the source client name of the given submit element.
572  **
573  *************************************************************************
                                                                        ********
574  */

576  int
577  GetSESourceClientName(int ID, int elementID, char *buff, int maxsize,
578                        int *status)
579 1 {
580 1     EDMRESubmitElement *se;
581 1     int ret;

583 1     if (status == NULL)
584 2     {
585 2         return -1;
586 1     }

588 1     if (buff == NULL)
589 2     {
590 2         *status = SUBMIT_BAD_PARAM;
591 2         return -1;
592 1     }

594 1     ret = LookupSubmitElement(ID, elementID, &se, status);

596 1     if (ret != 0)
597 1         return ret;

599 1     se -> getClientSource(buff, maxsize);

601 1     return 0;
602  }
```



```
604  /*************************************************************************
                                                                        ********
605  **
606  ** Routine:  GetSEDestClientName
607  **
608  ** Inputs:   int ID - submit ID associated with element
609  **           int elementID - submit element ID associated with element
610  **           int maxsize - maximum size to fill the buffer with
611  **
612  ** Outputs:  int *status - status of the function if an error occured
613  **           char *buff - place to put the client name
614  **
615  ** Return Codes:
616  **           int - 0 for success or non-zero for failure
617  **
618  ** Purpose:  Gets the destination client name of the given submit
                                                                        element.
619  **
620  *************************************************************************
                                                                        ********
621  */

623  int
624  GetSEDestClientName(int ID, int elementID, char *buff, int maxsize,
625                      int *status)
626 1 {
627 1     EDMRESubmitElement *se;
628 1     int ret;

630 1     if (status == NULL)
631 2     {
632 2         return -1;
633 1     }

635 1     if (buff == NULL)
636 2     {
637 2         *status = SUBMIT_BAD_PARAM;
638 2         return -1;
639 1     }

641 1     ret = LookupSubmitElement(ID, elementID, &se, status);

643 1     if (ret != 0)
644 1         return ret;

646 1     se -> getClientHostname(buff, maxsize);

648 1     return 0;
649  }
```

```
651  /*************************************************************************
                                                                        ********
652  **
653  ** Routine:  GetSEIsRestoreInplace
654  **
655  ** Inputs:   int ID - submit ID associated with element
656  **           int elementID - submit element ID associated with element
657  **
658  ** Outputs:  int *status - status of the function if an error occured
659  **
660  ** Return Codes:
661  **           boolean_ty - TRUE is yes and FALSE otherwise
662  **
663  ** Purpose:  Gets the Is-Restore-In-Place boolean from submit element.
664  **
665  *************************************************************************
                                                                        ********
666  */

668  boolean_ty
669  GetSEIsRestoreInplace(int ID, int elementID,
670                        int *status)
671 1 {
672 1     EDMRESubmitElement *se;
673 1     int ret;

675 1     if (status == NULL)
676 2     {
677 2         return FALSE;
678 1     }

680 1     ret = LookupSubmitElement(ID, elementID, &se, status);

682 1     if (ret != 0)
683 1         return FALSE;

685 1     return se -> IsInPlace();
686  }
```



```
688  /*************************************************************************
                                                                        ********
689  **
690  ** Routine:  GetSEDestDirTop
691  **
692  ** Inputs:   int ID - submit ID associated with element
693  **           int elementID - submit element ID associated with element
694  **           int maxsize - maximum size to fill the buffer with
695  **
696  ** Outputs:  int *status - status of the function if an error occured
697  **           char *buff - place to put the destination directory
698  **
699  ** Return Codes:
700  **           int - 0 for success or non-zero for failure
701  **
702  ** Purpose:  Gets the destination directory of the given submit
                                                                        element.
703  **
704  *************************************************************************
                                                                        ********
705  */

707  int
708  GetSEDestDirTop(int ID, int elementID, char *buff, int maxsize,
709                  int *status)
710 1 {
711 1     EDMRESubmitElement *se;
712 1     int ret;

714 1     if (status == NULL)
715 2     {
716 2         return -1;
717 1     }

719 1     if (buff == NULL)
720 2     {
721 2         *status = SUBMIT_BAD_PARAM;
722 2         return -1;
723 1     }

725 1     ret = LookupSubmitElement(ID, elementID, &se, status);

727 1     if (ret != 0)
728 1         return ret;

730 1     se -> getDirectoryTop(buff, maxsize);

732 1     return 0;
733  }
```

```
735  /*****************************************************************************
                                                                          ********
736  **
737  ** Routine:  GetSEDestDirectory
738  **
739  ** Inputs:   int ID - submit ID associated with element
740  **           int elementID - submit element ID associated with element
741  **           int maxsize - maximum size to fill the buffer with
742  **
743  ** Outputs:  int *status - status of the function if an error occured
744  **           char *buff - place to put the directory
745  **
746  ** Return Codes:
747  **           int - 0 for success or non-zero for failure
748  **
749  ** Purpose:  Gets the destination directory of the given submit
                                                                    element.
750  **
751  *****************************************************************************
                                                                          ********
752  */

754  int
755  GetSEDestDirectory(int ID, int elementID, char *buff, int maxsize,
756                     int *status)
757 1 {
758 1     EDMRESubmitElement *se;
759 1     int ret;

761 1     if (status == NULL)
762 2     {
763 2         return -1;
764 1     }

766 1     if (buff == NULL)
767 2     {
768 2         *status = SUBMIT_BAD_PARAM;
769 2         return -1;
770 1     }

772 1     ret = LookupSubmitElement(ID, elementID, &se, status);

774 1     if (ret != 0)
775 1         return ret;

777 1     se -> getDirectoryDestination(buff, maxsize);

779 1     return 0;
780  }
```



```
782  /*****************************************************************************
                                                                          ********
783  **
784  ** Routine:  GetSEDestOverWritePolicy
785  **
786  ** Inputs:   int ID - submit ID associated with element
787  **           int elementID - submit element ID associated with element
788  **
789  ** Outputs:  int *status - status of the function if an error occured
790  **
791  ** Return Codes:
792  **           OverwritePolicy
793  **
794  ** Purpose:  Gets the OverwritePolicy enum from submit element.
795  **
796  *****************************************************************************
                                                                          ********
797  */

799  OverwritePolicy
800  GetSEDestOverWritePolicy(int ID, int elementID,
801                           int *status)
802 1 {
803 1     EDMRESubmitElement *se;
804 1     int ret;

806 1     if (status == NULL)
807 2     {
808 2         return Older_Only_Overwrite;
                                        /* avoid warning: use default value */
809 1     }

811 1     ret = LookupSubmitElement(ID, elementID, &se, status);

813 1     if (ret != 0)
814 1         return Older_Only_Overwrite;
                                        /* avoid warning: use default value */

816 1     return se -> getOverwritePolicy();
817  }
```

```
819  /*************************************************************************
                                                                        ********
820  **
821  ** Routine:  GetSESubmitFile
822  **
823  ** Inputs:   int ID - submit ID associated with element
824  **           int elementID - submit element ID associated with element
825  **           int maxsize - maximum size to fill the buffer with
826  **
827  ** Outputs:  int *status - status of the function if an error occured
828  **           char *buff - place to put the submit file name
829  **
830  ** Return Codes:
831  **           int - 0 for success or non-zero for failure
832  **
833  ** Purpose:  Gets the submit file name of the given submit element.
834  **
835  *************************************************************************
                                                                        ********
836  */

838  int
839  GetSESubmitFile(int ID, int elementID, char *buff, int maxsize,
840                  int *status)
841 1 {
842 1     EDMRESubmitElement *se;
843 1     int ret;

845 1     if (status == NULL)
846 2     {
847 2         return -1;
848 1     }

850 1     if (buff == NULL)
851 2     {
852 2         *status = SUBMIT_BAD_PARAM;
853 2         return -1;
854 1     }

856 1     ret = LookupSubmitElement(ID, elementID, &se, status);

858 1     if (ret != 0)
859 1         return ret;

861 1     se -> getSubmitFile(buff, maxsize);

863 1     return 0;
864  }
```



```
866  /*************************************************************************
                                                                        ********
867  **
868  ** Routine:  GetSERcmdScriptName
869  **
870  ** Inputs:   int ID - submit ID associated with element
871  **           int elementID - submit element ID associated with element
872  **           int maxsize - maximum size to fill the buffer with
873  **
874  ** Outputs:  int *status - status of the function if an error occured
875  **           char *buff - place to put the client script name
876  **
877  ** Return Codes:
878  **           int - 0 for success or non-zero for failure
879  **
880  ** Purpose:  Gets the client script name of the given submit element.
881  **
882  *************************************************************************
                                                                        ********
883  */

885  int
886  GetSERcmdScriptName(int ID, int elementID, char *buff, int maxsize,
887                      int *status)
888 1 {
889 1     EDMRESubmitElement *se;
890 1     int ret;

892 1     if (status == NULL)
893 2     {
894 2         return -1;
895 1     }

897 1     if (buff == NULL)
898 2     {
899 2         *status = SUBMIT_BAD_PARAM;
900 2         return -1;
901 1     }

903 1     ret = LookupSubmitElement(ID, elementID, &se, status);

905 1     if (ret != 0)
906 1         return ret;

908 1     se -> getScriptName(buff, maxsize);

910 1     return 0;
911  }
```

```
913  /*************************************************************************
                                                                        ********
914  **
915  ** Routine:  GetSERcmdConnect
916  **
917  ** Inputs:   int ID - submit ID associated with element
918  **           int elementID - submit element ID associated with element
919  **           int maxsize - maximum size to fill the buffer with
920  **
921  ** Outputs:  int *status - status of the function if an error occured
922  **           char *buff - place to put the client name
923  **           int *port - place to put the port to connect on
924  **
925  ** Return Codes:
926  **           int - 0 for success or non-zero for failure
927  **
928  ** Purpose:  Gets the client name and port to connet on for the given
929  **           submit element.
930  **
931  *************************************************************************
                                                                        ********
932  */

934  int
935  GetSERcmdConnect(int ID, int elementID, char *buff, int maxsize,
936                   int *port, int *status)
937 1 {
938 1     EDMRESubmitElement *se;
939 1     int ret;

941 1     if (status == NULL)
942 2     {
943 2         return -1;
944 1     }

946 1     if (port == NULL || buff == NULL)
947 2     {
948 2         *status = SUBMIT_BAD_PARAM;
949 2         return -1;
950 1     }

952 1     ret = LookupSubmitElement(ID, elementID, &se, status);

954 1     if (ret != 0)
955 1         return ret;

957 1     se -> getSocketHost(buff, maxsize);
958 1     *port = se -> getSocketPort();

960 1     return 0;
961  }
```



```
963  /*************************************************************************
                                                                        ********
964  **
965  ** Routine:  GetSEMarkedSummary
966  **
967  ** Inputs:   int ID - submit ID associated with element
968  **           int elementID - submit element ID associated with element
969  **
970  ** Outputs:  int *status - status of the function if an error occured
971  **           struct mark_summary *buff - place to put the mark summary
972  **
973  ** Return Codes:
974  **           int - 0 for success or non-zero for failure
975  **
976  ** Purpose:  Gets the mark summary of the given submit element.
977  **
978  *************************************************************************
                                                                        ********
979  */

981  int
982  GetSEMarkedSummary(int ID, int elementID, struct mark_summary *buff,
983                     int *status)
984 1 {
985 1     EDMRESubmitElement *se;
986 1     int ret;

988 1     if (status == NULL)
989 2     {
990 2         return -1;
991 1     }

993 1     if (buff == NULL)
994 2     {
995 2         *status = SUBMIT_BAD_PARAM;
996 2         return -1;
997 1     }

999 1     ret = LookupSubmitElement(ID, elementID, &se, status);

1001 1    if (ret != 0)
1002 1        return ret;

1004 1    se -> getMarkSummary(buff);

1006 1    return 0;
1007 }
```

```
1009  /*****************************************************************
                                                                ********
1010  **
1011  ** Routine:  GetSEVolumesNeeded
1012  **
1013  ** Inputs:   int ID - submit ID associated with element
1014  **           int elementID - submit element ID associated with element
1015  **
1016  ** Outputs:  int *status - status of the function if an error occured
1017  **           ebvl_volidlist_ty **buff - place to put the vol list
1018  **
1019  ** Return Codes:
1020  **           int - 0 for success or non-zero for failure
1021  **
1022  ** Purpose:  Gets the vol list of the given submit element.
1023  **
1024  *****************************************************************
                                                                ********
1025  */

1027  int
1028  GetSEVolumesNeeded(int ID, int elementID, ebvl_volidlist_ty **buff,
1029                     int *status)
1030 1 {
1031 1     EDMRESubmitElement *se;
1032 1     int ret;

1034 1     if (status == NULL)
1035 2     {
1036 2         return -1;
1037 1     }

1039 1     if (buff == NULL)
1040 2     {
1041 2         *status = SUBMIT_BAD_PARAM;
1042 2         return -1;
1043 1     }

1045 1     ret = LookupSubmitElement(ID, elementID, &se, status);

1047 1     if (ret != 0)
1048 1         return ret;

1050 1     se -> getVolumeList(buff);

1052 1     return 0;
1053  }
```



```
1055  /*****************************************************************
                                                                ********
1056  **
1057  ** Routine:  GetSEWorkItemType
1058  **
1059  ** Inputs:   int ID - submit ID associated with element
1060  **           int elementID - submit element ID associated with element
1061  **
1062  ** Outputs:  int *status - status of the function if an error occured
1063  **           char *type - the work item type
1064  **
1065  ** Return Codes:
1066  **           int - 0 for success or non-zero for failure
1067  **
1068  ** Purpose:  Gets the vol list of the given submit element.
1069  **
1070  *****************************************************************
                                                                ********
1071  */

1073  int
1074  GetSEWorkItemType(int ID, int elementID, char *type,
1075                    int *status)
1076 1 {
1077 1     EDMRESubmitElement *se;
1078 1     int ret;

1080 1     if (status == NULL)
1081 2     {
1082 2         return -1;
1083 1     }

1085 1     if (type == NULL)
1086 2     {
1087 2         *status = SUBMIT_BAD_PARAM;
1088 2         return -1;
1089 1     }

1091 1     ret = LookupSubmitElement(ID, elementID, &se, status);

1093 1     if (ret != 0)
1094 1         return ret;

1096 1     *type = se -> getWorkItemType();

1098 1     return 0;
1099  }
```

```
1101  /*****************************************************************************
                                                                          ********
1102  **
1103  ** Routine:  GetSEPluginData
1104  **
1105  ** Inputs:   int ID - submit ID
1106  **           int elementID - submit element ID
1107  **
1108  ** Outputs:  int *status - a status of the operation
1109  **           RWCollection **plugin_data - ptr to plugin specific
                                                        submit element data
1110  **
1111  ** Return Codes:
1112  **           int - 0 for success and non-zero otherwise
1113  **
1114  ** Purpose:  Gets the pointer to the plugin specific data for the
                                                      submit element
1115  **
1116  *****************************************************************************
                                                                          ********
1117  */

1119  int
1120  GetSEPluginData(
          int ID, int elementID, RWCollection **plugin_data, int *status)
1121 1 {
1122 1     EDMRESubmitElement *se;
1123 1     int ret;

1125 1     if (status == NULL)
1126 2     {
1127 2         return -1;
1128 1     }

1130 1     if (plugin_data == NULL)
1131 2     {
1132 2         *status = SUBMIT_BAD_PARAM;
1133 2         return -1;
1134 1     }

1136 1     ret = LookupSubmitElement(ID, elementID, &se, status);

1138 1     if (ret != 0)
1139 1         return ret;

1141 1     *plugin_data = se -> getPluginData();

1143 1     return 0;
1144  }
```



```
1146  /*****************************************************************************
                                                                          ********
1147  **
1148  ** Routine:  GetSOBackupAdmin
1149  **
1150  ** Inputs:   int ID - submit ID associated with element
1151  **
1152  ** Outputs:  int *status - status of the function if an error occured
1153  **
1154  ** Return Codes:
1155  **           boolean_ty - TRUE if the user is and FALSE otherwise.
1156  **
1157  ** Purpose:  Determine whether the user is backup admin.
1158  **
1159  *****************************************************************************
                                                                          ********
1160  */

1162  boolean_ty
1163  GetSOBackupAdmin(int ID, int *status)
1164 1 {
1165 1     EDMRESubmitObj *so;
1166 1     int ret;

1168 1     if (status == NULL)
1169 2     {
1170 2         return FALSE;
1171 1     }

1173 1     ret = LookupSubmitObject(ID, &so, status);

1175 1     if (ret != 0)
1176 1         return FALSE;

1178 1     return so -> IsBackupAdmin();
1179  }
```

```
1181  /*****************************************************************************
                                                                          ********
1182  **
1183  ** Routine:  GetSOSourceSystemAdmin
1184  **
1185  ** Inputs:   int ID - submit ID associated with element
1186  **
1187  ** Outputs:  int *status - status of the function if an error occured
1188  **
1189  ** Return Codes:
1190  **           boolean_ty - TRUE if the user is and FALSE otherwise.
1191  **
1192  ** Purpose:  Determine whether the user is source system admin.
1193  **
1194  *****************************************************************************
                                                                          ********
1195  */

1197  boolean_ty
1198  GetSOSourceSystemAdmin(int ID, int *status)
1199 1 {
1200 1     EDMRESubmitObj *so;
1201 1     int ret;

1203 1     if (status == NULL)
1204 2     {
1205 2         return FALSE;
1206 1     }

1208 1     ret = LookupSubmitObject(ID, &so, status);

1210 1     if (ret != 0)
1211 1         return FALSE;

1213 1     return so -> IsSourceSystemAdmin();
1214  }
```



```
1216  /*****************************************************************************
                                                                          ********
1217  **
1218  ** Routine:  GetSODestinationAdmin
1219  **
1220  ** Inputs:   int ID - submit ID associated with element
1221  **
1222  ** Outputs:  int *status - status of the function if an error occured
1223  **
1224  ** Return Codes:
1225  **           boolean_ty - TRUE if the user is and FALSE otherwise.
1226  **
1227  ** Purpose:  Determine whether the user is destination system admin.
1228  **
1229  *****************************************************************************
                                                                          ********
1230  */

1232  boolean_ty
1233  GetSODestinationAdmin(int ID, int *status)
1234 1 {
1235 1     EDMRESubmitObj *so;
1236 1     int ret;

1238 1     if (status == NULL)
1239 2     {
1240 2         return FALSE;
1241 1     }

1243 1     ret = LookupSubmitObject(ID, &so, status);

1245 1     if (ret != 0)
1246 1         return FALSE;

1248 1     return so -> IsDestinationAdmin();
1249  }
```

```
1251  /*****************************************************************************
                                                                          ********
1252  **
1253  ** Routine:  GetSOUserID
1254  **
1255  ** Inputs:   int ID - submit ID associated with element
1256  **
1257  ** Outputs:  int *status - status of the function if an error occured
1258  **           uid_t *userid - place to put the user ID
1259  **
1260  ** Return Codes:
1261  **           int - 0 if success and non-zero otherwise
1262  **
1263  ** Purpose:  Get the user ID.
1264  **
1265  *****************************************************************************
                                                                          ********
1266  */

1268  int
1269  GetSOUserID(int ID, uid_t *userid, int *status)
1270  1 {
1271  1     EDMRESubmitObj *so;
1272  1     int ret;

1274  1     if (status == NULL)
1275  2     {
1276  2         return -1;
1277  1     }

1279  1     if (userid == NULL)
1280  2     {
1281  2         *status = SUBMIT_BAD_PARAM;
1282  2         return -1;
1283  1     }

1285  1     ret = LookupSubmitObject(ID, &so, status);

1287  1     if (ret != 0)
1288  2     {
1289  2         return ret;
1290  1     }

1292  1     *userid = so -> getUserID();

1294  1     return 0;
1295  }
```



```
1297  /*****************************************************************************
                                                                          ********
1298  **
1299  ** Routine:  GetSOEffectiveUID
1300  **
1301  ** Inputs:   int ID - submit ID
1302  **
1303  ** Outputs:  int *status - status of the function if an error occured
1304  **           uid_t *userid - place to put the user ID
1305  **
1306  ** Return Codes:
1307  **           int - 0 if success and non-zero otherwise
1308  **
1309  ** Purpose:  Get the effective user ID.
1310  **
1311  *****************************************************************************
                                                                          ********
1312  */

1314  int
1315  GetSOEffectiveUID(int ID, uid_t *userid, int *status)
1316  1 {
1317  1     EDMRESubmitObj *so;
1318  1     int ret;

1320  1     if (status == NULL)
1321  2     {
1322  2         return -1;
1323  1     }

1325  1     if (userid == NULL)
1326  2     {
1327  2         *status = SUBMIT_BAD_PARAM;
1328  2         return -1;
1329  1     }

1331  1     ret = LookupSubmitObject(ID, &so, status);

1333  1     if (ret != 0)
1334  2     {
1335  2         return ret;
1336  1     }

1338  1     *userid = so -> getEffectiveUserID();

1340  1     return 0;
1341  }
```

```
1343  /*************************************************************************
                                                                        ********
1344  **
1345  ** Routine:  GetSOUserName
1346  **
1347  ** Inputs:   int ID - submit ID
1348  **           int maxsize - maximum size to fill the buffer with
1349  **
1350  ** Outputs:  int *status - status of the function if an error occured
1351  **           char *buff - place to put the user name
1352  **
1353  ** Return Codes:
1354  **           int - 0 for success or non-zero for failure
1355  **
1356  ** Purpose:  Gets the user name to be used in the restore.
1357  **
1358  *************************************************************************
                                                                        ********
1359  */

1361  int
1362  GetSOUserName(int ID, char *buff, int maxsize, int *status)
1363 1 {
1364 1     EDMRESubmitObj *so;
1365 1     int ret;

1367 1     if (status == NULL)
1368 2     {
1369 2         return -1;
1370 1     }

1372 1     if (buff == NULL)
1373 2     {
1374 2         *status = SUBMIT_BAD_PARAM;
1375 2         return -1;
1376 1     }

1378 1     ret = LookupSubmitObject(ID, &so, status);

1380 1     if (ret != 0)
1381 2     {
1382 2         return ret;
1383 1     }

1385 1     so -> getUserName(buff, maxsize);

1387 1     return 0;
1388  }
```



```
1390  /*************************************************************************
                                                                        ********
1391  **
1392  ** Routine:  GetSOEffectiveUserName
1393  **
1394  ** Inputs:   int ID - submit ID
1395  **           int maxsize - maximum size to fill the buffer with
1396  **
1397  ** Outputs:  int *status - status of the function if an error occured
1398  **           char *buff - place to put the effective user name
1399  **
1400  ** Return Codes:
1401  **           int - 0 for success or non-zero for failure
1402  **
1403  ** Purpose:  Gets the effective user name to be used in the restore.
1404  **
1405  *************************************************************************
                                                                        ********
1406  */

1408  int
1409  GetSOEffectiveUserName(int ID, char *buff, int maxsize, int *status)
1410 1 {
1411 1     EDMRESubmitObj *so;
1412 1     int ret;

1414 1     if (status == NULL)
1415 2     {
1416 2         return -1;
1417 1     }

1419 1     if (buff == NULL)
1420 2     {
1421 2         *status = SUBMIT_BAD_PARAM;
1422 2         return -1;
1423 1     }

1425 1     ret = LookupSubmitObject(ID, &so, status);

1427 1     if (ret != 0)
1428 2     {
1429 2         return ret;
1430 1     }

1432 1     so -> getEffectiveUserName(buff, maxsize);

1434 1     return 0;
1435  }
```

```
1437  /*****************************************************************************
                                                                          ********
1438  **
1439  ** Routine:  GetSOMarkedSummary
1440  **
1441  ** Inputs:   int ID - submit ID
1442  **
1443  ** Outputs:  int *status - status of the function if an error occured
1444  **           struct mark_summary *buff - place to put the mark summary
1445  **
1446  ** Return Codes:
1447  **           int - 0 for success or non-zero for failure
1448  **
1449  ** Purpose:  Gets the mark summary of the given submit object.
1450  **
1451  *****************************************************************************
                                                                          ********
1452  */

1454  int
1455  GetSOMarkedSummary(int ID, struct mark_summary *buff, int *status)
1456 1 {
1457 1     EDMRESubmitObj *so;
1458 1     int ret;

1460 1     if (status == NULL)
1461 2     {
1462 2         return -1;
1463 1     }

1465 1     if (buff == NULL)
1466 2     {
1467 2         *status = SUBMIT_BAD_PARAM;
1468 2         return -1;
1469 1     }

1471 1     ret = LookupSubmitObject(ID, &so, status);

1473 1     if (ret != 0)
1474 1         return ret;

1476 1     so -> getMarkSummary(buff);

1478 1     return 0;
1479  }
```



```
1481  /*****************************************************************************
                                                                          ********
1482  **
1483  ** Routine:  GetSOVolumesNeeded
1484  **
1485  ** Inputs:   int ID - submit ID
1486  **           int elementID - submit element ID associated with element
1487  **
1488  ** Outputs:  int *status - status of the function if an error occured
1489  **           ebvl_volidlist_ty **buff - place to put the vol list
1490  **
1491  ** Return Codes:
1492  **           int - 0 for success or non-zero for failure
1493  **
1494  ** Purpose:  Gets the vol list of the given submit element.
1495  **
1496  *****************************************************************************
                                                                          ********
1497  */

1499  int
1500  GetSOVolumesNeeded(int ID, ebvl_volidlist_ty **buff, int *status)
1501 1 {
1502 1     EDMRESubmitObj *so;
1503 1     int ret;

1505 1     if (status == NULL)
1506 2     {
1507 2         return -1;
1508 1     }

1510 1     if (buff == NULL)
1511 2     {
1512 2         *status = SUBMIT_BAD_PARAM;
1513 2         return -1;
1514 1     }

1516 1     ret = LookupSubmitObject(ID, &so, status);

1518 1     if (ret != 0)
1519 1         return ret;

1521 1     so -> getVolumeList(buff);

1523 1     return 0;
1524  }
```

```
1526  /*************************************************************************
                                                                        ********
1527  **
1528  ** Routine:  GetSOWorkItemCount
1529  **
1530  ** Inputs:   int ID - submit ID
1531  **
1532  ** Outputs:  int *status - status of the function if an error occured
1533  **           int *wicount - place to put the work item count
1534  **
1535  ** Return Codes:
1536  **           int - 0 if success and non-zero otherwise
1537  **
1538  ** Purpose:  Get the user ID.
1539  **
1540  *************************************************************************
                                                                        ********
1541  */

1543  int
1544  GetSOWorkItemCount(int ID, int *wicount, int *status)
1545  1 {
1546  1     EDMRESubmitObj *so;
1547  1     int ret;

1549  1     if (status == NULL)
1550  2     {
1551  2         return -1;
1552  1     }

1554  1     if (wicount == NULL)
1555  2     {
1556  2         *status = SUBMIT_BAD_PARAM;
1557  2         return -1;
1558  1     }

1560  1     ret = LookupSubmitObject(ID, &so, status);

1562  1     if (ret != 0)
1563  2     {
1564  2         return ret;
1565  1     }

1567  1     *wicount = so -> getWICount();

1569  1     return 0;
1570  }
```



```
1572  /*************************************************************************
                                                                        ********
1573  **
1574  ** Routine:  GetSOPrePhase
1575  **
1576  ** Inputs:   int ID - submit ID
1577  **           int maxsize - maximum size to fill the executable buffer
                                                                          with
1578  **
1579  ** Outputs:  int *status - status of the function if an error occured
1580  **           char **executable - place to put the pre-phase executable
                                                                          name
1581  **           char ***prephaseargs - place to put the pre-phase
                                                                      arguments
1582  **           char ***prephaseenv - place to put the pre-phase
                                                            environment vars
1583  **
1584  ** Return Codes:
1585  **           int - 0 for success or non-zero for failure
1586  **
1587  ** Purpose:  Gets all pertinent information regarding the pre-phase
                                                                     runtime.
1588  **
1589  *************************************************************************
                                                                        ********
1590  */

1592  int
1593  GetSOPrePhase(int ID, char *executable, int maxsize,
1594                char ***prephaseargs, char ***prephaseenv, int *status)
1595  1 {
1596  1     EDMRESubmitObj *so;
1597  1     int ret;

1599  1     if (status == NULL)
1600  2     {
1601  2         return -1;
1602  1     }

1604  1     if (executable == NULL || prephaseargs == NULL || prephaseenv ==
                                                                        NULL)
1605  2     {
1606  2         *status = SUBMIT_BAD_PARAM;
1607  2         return -1;
1608  1     }

1610  1     ret = LookupSubmitObject(ID, &so, status);

1612  1     if (ret != 0)
1613  1         return ret;

1615  1     so -> getPrePhaseExecutable(executable, maxsize);

1617  1     so -> getPrePhaseArgs(prephaseargs);

1619  1     so -> getPrePhaseEnv(prephaseenv);

1621  1     return 0;
1622  }
```

```
1624  /*************************************************************************
                                                                        ********
1625  **
1626  ** Routine:  GetSOExecutionPhase
1627  **
1628  ** Inputs:   int ID - submit ID
1629  **           int maxsize - maximum size to fill the executable buffer
                                                                           with
1630  **
1631  ** Outputs:  int *status - status of the function if an error occured
1632  **           char **executable - place to put the execution-phase
                                                              executable name
1633  **           char ***executionphaseargs - place to put the
                                                    execution-phase arguments
1634  **           char ***executionphaseenv - place to put the
                                               execution-phase environment vars
1635  **
1636  ** Return Codes:
1637  **           int - 0 for success or non-zero for failure
1638  **
1639  ** Purpose:  Gets all pertinent information regarding the
                                                    execution-phase runtime.
1640  **
1641  *************************************************************************
                                                                        ********
1642  */

1644  int
1645  GetSOExecutionPhase(int ID, char *executable, int maxsize,
1646                char ***executionphaseargs, char ***executionphaseenv,
                                                                  int *status)
1647 1 {
1648 1     EDMRESubmitObj *so;
1649 1     int ret;

1651 1     if (status == NULL)
1652 2     {
1653 2         return -1;
1654 1     }

1656 1     if (executable == NULL || executionphaseargs == NULL ||
1657 1         executionphaseenv == NULL)
1658 2     {
1659 2         *status = SUBMIT_BAD_PARAM;
1660 2         return -1;
1661 1     }

1663 1     ret = LookupSubmitObject(ID, &so, status);

1665 1     if (ret != 0)
1666 1         return ret;

1668 1     so -> getExecutePhaseExecutable(executable, maxsize);

1670 1     so -> getExecutePhaseArgs(executionphaseargs);

1672 1     so -> getExecutePhaseEnv(executionphaseenv);

1674 1     return 0;
1675  }
```



```
1677  /*************************************************************************
                                                                        ********
1678  **
1679  ** Routine:  GetSOPostPhase
1680  **
1681  ** Inputs:   int ID - submit ID
1682  **           int maxsize - maximum size to fill the executable buffer
                                                                           with
1683  **
1684  ** Outputs:  int *status - status of the function if an error occured
1685  **           char **executable - place to put the post-phase
                                                              executable name
1686  **           char ***postphaseargs - place to put the post-phase
                                                                   arguments
1687  **           char ***postphaseenv - place to put the post-phase
                                                               environment vars
1688  **
1689  ** Return Codes:
1690  **           int - 0 for success or non-zero for failure
1691  **
1692  ** Purpose:  Gets all pertinent information regarding the post-phase
                                                                     runtime.
1693  **
1694  *************************************************************************
                                                                        ********
1695  */

1697  int
1698  GetSOPostPhase(int ID, char *executable, int maxsize,
1699                char ***postphaseargs, char ***postphaseenv,
                                                                  int *status)
1700 1 {
1701 1     EDMRESubmitObj *so;
1702 1     int ret;

1704 1     if (status == NULL)
1705 2     {
1706 2         return -1;
1707 1     }

1709 1     if (executable == NULL || postphaseargs == NULL || postphaseenv ==
                                                                        NULL)
1710 2     {
1711 2         *status = SUBMIT_BAD_PARAM;
1712 2         return -1;
1713 1     }

1715 1     ret = LookupSubmitObject(ID, &so, status);

1717 1     if (ret != 0)
1718 1         return ret;

1720 1     so -> getPostPhaseExecutable(executable, maxsize);

1722 1     so -> getPostPhaseArgs(postphaseargs);

1724 1     so -> getPostPhaseEnv(postphaseenv);

1726 1     return 0;
1727  }
```

```
1729  /*************************************************************************
                                                                        ********
1730  **
1731  ** Routine:  IsValidSubmitID
1732  **
1733  ** Inputs:   int ID - submit ID
1734  **
1735  ** Outputs:  None
1736  **
1737  ** Return Codes:
1738  **           boolean_ty - TRUE a good ID and FALSE otherwise.
1739  **
1740  ** Purpose:  Returns whether the submit ID is good or not.
1741  **
1742  *************************************************************************
                                                                        ********
1743  */

1745  boolean_ty
1746  IsValidSubmitID(int ID)
1747 1 {
1748 1     EDMRESubmitObj *so;
1749 1     int ret, status;

1751 1     if (ID == 0)
1752 1         return FALSE;

1754 1     ret = LookupSubmitObject(ID, &so, &status);

1756 1     if (ret != 0)
1757 2     {
1758 2         return FALSE;
1759 1     }

1761 1     return TRUE;
1762   }
```



```
1764  /*************************************************************************
                                                                        ********
1765  **
1766  ** Routine:  SetSOBasics
1767  **
1768  ** Inputs:   int ID - submit ID
1769  **           unsigned int type - restore type
1770  **           unsigned int flags - execution flags
1771  **
1772  ** Outputs:  int *status - a status of the operation
1773  **
1774  ** Return Codes:
1775  **           int - 0 for success and non-zero otherwise
1776  **
1777  ** Purpose:  Sets some submit object internals.
1778  **
1779  *************************************************************************
                                                                        ********
1780  */

1782  int
1783  SetSOBasics(int ID, unsigned int type, unsigned int flags,
1784                          int *status)
1785 1 {
1786 1     EDMRESubmitObj *so;
1787 1     int ret;

1789 1     if (status == NULL)
1790 2     {
1791 2         return -1;
1792 1     }

1794 1     ret = LookupSubmitObject(ID, &so, status);

1796 1     if (ret != 0)
1797 2     {
1798 2         return ret;
1799 1     }

1801 1     so -> setSubmitType(type);
1802 1     so -> setExecutionFlags(flags);

1804 1     return 0;
1805   }
```

```
1807  /*****************************************************************************
                                                                          ********
1808  **
1809  ** Routine:  SetSOVMCheck
1810  **
1811  ** Inputs:   int ID - submit ID
1812  **           unsigned int type - restore type
1813  **           unsigned int flags - execution flags
1814  **
1815  ** Outputs:  int *status - a status of the operation
1816  **
1817  ** Return Codes:
1818  **           int - 0 for success and non-zero otherwise
1819  **
1820  ** Purpose:  Sets vm check variable.
1821  **
1822  *****************************************************************************
                                                                          ********
1823  */

1825  int
1826  SetSOVMCheck(int ID, boolean_ty vmcheck, int *status)
1827 1 {
1828 1     EDMRESubmitObj *so;
1829 1     int ret;

1831 1     if (status == NULL)
1832 2     {
1833 2         return -1;
1834 1     }

1836 1     ret = LookupSubmitObject(ID, &so, status);

1838 1     if (ret != 0)
1839 2     {
1840 2         return ret;
1841 1     }

1843 1     so -> setNoVMCheck(vmcheck);

1845 1     return 0;
1846  }
```



```
1848  /*****************************************************************************
                                                                          ********
1849  **
1850  ** Routine:  SetSOPrePhase
1851  **
1852  ** Inputs:   int ID - submit ID
1853  **           char *executable - executable to run in the pre-phase
1854  **           char **args - args to use in the pre-phase
1855  **           char **env - environment to use in the pre-phase
1856  **
1857  ** Outputs:  int *status - a status of the operation
1858  **
1859  ** Return Codes:
1860  **           int - 0 for success and non-zero otherwise
1861  **
1862  ** Purpose:  Sets pre-phase variables.
1863  **
1864  *****************************************************************************
                                                                          ********
1865  */

1867  int
1868  SetSOPrePhase(int ID, char *executable, char **args, char **env,
1869                int *status)
1870 1 {
1871 1     EDMRESubmitObj *so;
1872 1     int ret;

1874 1     if (status == NULL)
1875 2     {
1876 2         return -1;
1877 1     }

1879 1     if (executable == NULL)
1880 2     {
1881 2         *status = SUBMIT_BAD_PARAM;
1882 2         return -1;
1883 1     }

1885 1     ret = LookupSubmitObject(ID, &so, status);

1887 1     if (ret != 0)
1888 2     {
1889 2         return ret;
1890 1     }

1892 1     so -> setPrePhaseExecutable(executable);

1894 1     if (args != NULL)
1895 1         so -> setPrePhaseArgs(args);

1897 1     if (env != NULL)
1898 1         so -> setPrePhaseEnv(env);

1900 1     return 0;
1901  }
```

```
1903  /*************************************************************************
                                                                        ********
1904  **
1905  ** Routine:  SetSOExecutePhase
1906  **
1907  ** Inputs:   int ID - submit ID
1908  **           char *executable - executable to run in the execute-phase
1909  **           char **args - args to use in the execute-phase
1910  **           char **env - environment to use in the execute-phase
1911  **
1912  ** Outputs:  int *status - a status of the operation
1913  **
1914  ** Return Codes:
1915  **           int - 0 for success and non-zero otherwise
1916  **
1917  ** Purpose:  Sets execute-phase variables.
1918  **
1919  *************************************************************************
                                                                        ********
1920  */

1922  int
1923  SetSOExecutePhase(int ID, char *executable, char **args, char **env,
1924                    int *status)
1925 1 {
1926 1     EDMRESubmitObj *so;
1927 1     int ret;

1929 1     if (status == NULL)
1930 2     {
1931 2         return -1;
1932 1     }

1934 1     if (executable == NULL && args == NULL && env == NULL)
1935 2     {
1936 2         *status = SUBMIT_BAD_PARAM;
1937 2         return -1;
1938 1     }

1940 1     ret = LookupSubmitObject(ID, &so, status);

1942 1     if (ret != 0)
1943 2     {
1944 2         return ret;
1945 1     }

1947 1     if (executable != NULL)
1948 1         so -> setExecutePhaseExecutable(executable);

1950 1     if (args != NULL)
1951 1         so -> setExecutePhaseArgs(args);

1953 1     if (env != NULL)
1954 1         so -> setExecutePhaseEnv(env);

1956 1     return 0;
1957  }
```



```
1959  /*************************************************************************
                                                                        ********
1960  **
1961  ** Routine:  SetSOPostPhase
1962  **
1963  ** Inputs:   int ID - submit ID
1964  **           char *executable - executable to run in the post-phase
1965  **           char **args - args to use in the post-phase
1966  **           char **env - environment to use in the post-phase
1967  **
1968  ** Outputs:  int *status - a status of the operation
1969  **
1970  ** Return Codes:
1971  **           int - 0 for success and non-zero otherwise
1972  **
1973  ** Purpose:  Sets post-phase variables.
1974  **
1975  *************************************************************************
                                                                        ********
1976  */

1978  int
1979  SetSOPostPhase(int ID, char *executable, char **args, char **env,
1980                 int *status)
1981 1 {
1982 1     EDMRESubmitObj *so;
1983 1     int ret;

1985 1     if (status == NULL)
1986 2     {
1987 2         return -1;
1988 1     }

1990 1     if (executable == NULL)
1991 2     {
1992 2         *status = SUBMIT_BAD_PARAM;
1993 2         return -1;
1994 1     }

1996 1     ret = LookupSubmitObject(ID, &so, status);

1998 1     if (ret != 0)
1999 2     {
2000 2         return ret;
2001 1     }

2003 1     so -> setPostPhaseExecutable(executable);

2005 1     if (args != NULL)
2006 1         so -> setPostPhaseArgs(args);

2008 1     if (env != NULL)
2009 1         so -> setPostPhaseEnv(env);

2011 1     return 0;
2012  }
```

```
2014  /*****************************************************************************
                                                                          ********
2015  **
2016  ** Routine:  SetSOAdminID
2017  **
2018  ** Inputs:   int ID - submit ID
2019  **           boolean_ty isbackupadmin - is the user backup admin
2020  **           boolean_ty issrcadmin - is the admin of the source client
2021  **           boolean_ty isdestadmin - is the admin of the destination
                                                                    client
2022  **
2023  ** Outputs:  int *status - a status of the operation
2024  **
2025  ** Return Codes:
2026  **           int - 0 for success and non-zero otherwise
2027  **
2028  ** Purpose:  Sets all admin booleans.
2029  **
2030  *****************************************************************************
                                                                          ********
2031  */

2033  int
2034  SetSOAdminID(int ID, boolean_ty isbackupadmin, boolean_ty issrcadmin,
2035                       boolean_ty isdestadmin, int *status)
2036 1 {
2037 1     EDMRESubmitObj *so;
2038 1     int ret;

2040 1     if (status == NULL)
2041 2     {
2042 2         return -1;
2043 1     }

2045 1     ret = LookupSubmitObject(ID, &so, status);

2047 1     if (ret != 0)
2048 2     {
2049 2         return ret;
2050 1     }

2052 1     so -> setIsBackupAdmin(isbackupadmin);
2053 1     so -> setIsSourceSystemAdmin(issrcadmin);
2054 1     so -> setIsDestinationAdmin(isdestadmin);

2056 1     return 0;
2057  }
```



```
2059  /*****************************************************************************
                                                                          ********
2060  **
2061  ** Routine:  SetSOUserID
2062  **
2063  ** Inputs:   int ID - submit ID
2064  **           uid_t userid - is the user ID of the restore user
2065  **           char *username - is the user name of the restore user
2066  **           uid_t euid - is the effective user ID of the restore user
2067  **           char *eusername - is the effective user name of the
                                                                    restore user
2068  **
2069  ** Outputs:  int *status - a status of the operation
2070  **
2071  ** Return Codes:
2072  **           int - 0 for success and non-zero otherwise
2073  **
2074  ** Purpose:  Sets all user ID variables.
2075  **
2076  *****************************************************************************
                                                                          ********
2077  */

2079  int
2080  SetSOUserID(int ID, uid_t userid, char *username, uid_t euid,
2081                      char *eusername, int *status)
2082 1 {
2083 1     EDMRESubmitObj *so;
2084 1     int ret;

2086 1     if (status == NULL)
2087 2     {
2088 2         return -1;
2089 1     }

2091 1     if (username == NULL || eusername == NULL)
2092 2     {
2093 2         *status = SUBMIT_BAD_PARAM;
2094 2         return -1;
2095 1     }

2097 1     ret = LookupSubmitObject(ID, &so, status);

2099 1     if (ret != 0)
2100 2     {
2101 2         return ret;
2102 1     }

2104 1     so -> setUserID(userid);
2105 1     so -> setEffectiveUserID(euid);
2106 1     so -> setUserName(username);
2107 1     so -> setEffectiveUserName(eusername);

2109 1     return 0;
2110  }
```

```
2112  /*************************************************************************
                                                                        ********
2113  **
2114  ** Routine:  SetSOTotalSize
2115  **
2116  ** Inputs:   int ID - submit ID
2117  **           struct mark_summary *markptr - the sum of all marks
2118  **
2119  ** Outputs:  int *status - a status of the operation
2120  **
2121  ** Return Codes:
2122  **           int - 0 for success and non-zero otherwise
2123  **
2124  ** Purpose:  Sets the internal pointer to the sum of the mark pointers
2125  **
2126  *************************************************************************
                                                                        ********
2127  */

2129  int
2130  SetSOTotalSize(int ID, struct mark_summary *markptr, int *status)
2131 1 {
2132 1     EDMRESubmitObj *so;
2133 1     int ret;

2135 1     if (status == NULL)
2136 2     {
2137 2         return -1;
2138 1     }

2140 1     if (markptr == NULL)
2141 2     {
2142 2         *status = SUBMIT_BAD_PARAM;
2143 2         return -1;
2144 1     }

2146 1     ret = LookupSubmitObject(ID, &so, status);

2148 1     if (ret != 0)
2149 2     {
2150 2         return ret;
2151 1     }

2153 1     so -> setMarkSummary(markptr);

2155 1     return 0;
2156  }
```



```
2158  /*************************************************************************
                                                                        ********
2159  **
2160  ** Routine:  SetSOTotalVolumes
2161  **
2162  ** Inputs:   int ID - submit ID
2163  **           ebvl_volidlist_ty *volptr - the sum of all volumes
2164  **
2165  ** Outputs:  int *status - a status of the operation
2166  **
2167  ** Return Codes:
2168  **           int - 0 for success and non-zero otherwise
2169  **
2170  ** Purpose:  Sets all admin booleans.
2171  **
2172  *************************************************************************
                                                                        ********
2173  */

2175  int
2176  SetSOTotalVolumes(int ID, ebvl_volidlist_ty *volptr, int *status)
2177 1 {
2178 1     EDMRESubmitObj *so;
2179 1     int ret;

2181 1     if (status == NULL)
2182 2     {
2183 2         return -1;
2184 1     }

2186 1     if (volptr == NULL)
2187 2     {
2188 2         *status = SUBMIT_BAD_PARAM;
2189 2         return -1;
2190 1     }

2192 1     ret = LookupSubmitObject(ID, &so, status);

2194 1     if (ret != 0)
2195 2     {
2196 2         return ret;
2197 1     }

2199 1     so -> setVolumeList(volptr);

2201 1     return 0;
2202  }
```

```
2204  /*************************************************************************
                                                                        ********
2205  **
2206  ** Routine:  SetSEBasics
2207  **
2208  ** Inputs:   int ID - submit ID
2209  **           int elementID - submit element ID
2210  **           char *workitem - the work item of the submit element
2211  **           char *templatename - the template of the submit element
2212  **           boolean_ty istrailsetalt - whether the trailset of the
2213  **                                      submit element is the alternate one
2214  **           char *clienthost - the source host of the restore for the
2215  **                              submit element
2216  **           char type - the type of the work item (
                                                    KICKER, LISTENER, ...)
2217  **
2218  ** Outputs:  int *status - a status of the operation
2219  **
2220  ** Return Codes:
2221  **           int - 0 for success and non-zero otherwise
2222  **
2223  ** Purpose:  Sets all submit element basics.
2224  **
2225  *************************************************************************
                                                                        ********
2226  */

2228  int
2229  SetSEBasics(int ID, int elementID, char *workitem,
2230              char *templatename, boolean_ty istrailsetalt,
2231              char *clienthost, char type, int *status)
2232 1 {
2233 1     EDMRESubmitElement *se;
2234 1     int ret;

2236 1     if (status == NULL)
2237 2     {
2238 2         return -1;
2239 1     }

2241 1     if (workitem == NULL || templatename == NULL || clienthost ==
                                                                        NULL)
2242 2     {
2243 2         *status = SUBMIT_BAD_PARAM;
2244 2         return -1;
2245 1     }

2247 1     ret = LookupSubmitElement(ID, elementID, &se, status);

2249 1     if (ret != 0)
2250 2     {
2251 2         return ret;
2252 1     }

2254 1     se -> setWorkItem(workitem);
2255 1     se -> setTemplate(templatename);
2256 1     se -> SetAlternateTrailset(istrailsetalt);
2257 1     se -> setClientSource(clienthost);
2258 1     se -> setWorkItemType(type);

2260 1     return 0;
2261  }
```



```
2263  /*************************************************************************
                                                                        ********
2264  **
2265  ** Routine:  SetSEDestination
2266  **
2267  ** Inputs:   int ID - submit ID
2268  **           int elementID - submit element ID
2269  **           char *clientname - the destination client of the submit
                                                                        element
2270  **           boolean_ty inplace - whether the restore is in-place or
                                                                        not
2271  **           char *directory - the directory to restore to
2272  **           OverwritePolicy policy - the policy of this restore
2273  **           RestoreTransport transpt - the transport of this restore
2274  **
2275  ** Outputs:  int *status - a status of the operation
2276  **
2277  ** Return Codes:
2278  **           int - 0 for success and non-zero otherwise
2279  **
2280  ** Purpose:  Sets all submit element destination variables.
2281  **
2282  *************************************************************************
                                                                        ********
2283  */

2285  int
2286  SetSEDestination(int ID, int elementID, char *clientname,
2287              boolean_ty inplace, char *directory,
2288              OverwritePolicy policy, RestoreTransport transpt,
                                                                int *status)
2289 1 {
2290 1     EDMRESubmitElement *se;
2291 1     int ret;

2293 1     if (status == NULL)
2294 2     {
2295 2         return -1;
2296 1     }

2298 1     if (clientname == NULL || directory == NULL)
2299 2     {
2300 2         *status = SUBMIT_BAD_PARAM;
2301 2         return -1;
2302 1     }

2304 1     ret = LookupSubmitElement(ID, elementID, &se, status);

2306 1     if (ret != 0)
2307 2     {
2308 2         return ret;
2309 1     }

2311 1     se -> setClientHostname(clientname);
2312 1     se -> setInPlace(inplace);
2313 1     se -> setDirectoryDestination(directory);
2314 1     se -> setOverwritePolicy(policy);
2315 1     se -> setRestoreTransport(transpt);

2317 1     return 0;
2318  }
```

```
2320  /*****************************************************************************
                                                                          ********
2321  **
2322  ** Routine:  SetSEDirtop
2323  **
2324  ** Inputs:   int ID - submit ID
2325  **           int elementID - submit element ID
2326  **           char *dirtop - the top of the directory tree for the
                                                              submit element
2327  **
2328  ** Outputs:  int *status - a status of the operation
2329  **
2330  ** Return Codes:
2331  **           int - 0 for success and non-zero otherwise
2332  **
2333  ** Purpose:  Sets the top of the directory tree for the submit element.
2334  **
2335  *****************************************************************************
                                                                          ********
2336  */

2338  int
2339  SetSEDirtop(int ID, int elementID, char *dirtop, int *status)
2340 1 {
2341 1     EDMRESubmitElement *se;
2342 1     int ret;

2344 1     if (status == NULL)
2345 2     {
2346 2         return -1;
2347 1     }

2349 1     if (dirtop == NULL)
2350 2     {
2351 2         *status = SUBMIT_BAD_PARAM;
2352 2         return -1;
2353 1     }

2355 1     ret = LookupSubmitElement(ID, elementID, &se, status);

2357 1     if (ret != 0)
2358 2     {
2359 2         return ret;
2360 1     }

2362 1     se -> setDirectoryTop(dirtop);

2364 1     return 0;
2365  }
```



```
2367  /*****************************************************************************
                                                                          ********
2368  **
2369  ** Routine:  SetSEScriptName
2370  **
2371  ** Inputs:   int ID - submit ID
2372  **           int elementID - submit element ID
2373  **           char *scriptname - the script name to run on the client
2374  **           char *username - the user name to use when running the
                                                                        script
2375  **
2376  ** Outputs:  int *status - a status of the operation
2377  **
2378  ** Return Codes:
2379  **           int - 0 for success and non-zero otherwise
2380  **
2381  ** Purpose:  Sets the script name and username to run on the client.
2382  **
2383  *****************************************************************************
                                                                          ********
2384  */

2386  int
2387  SetSEScriptName(int ID, int elementID, char *scriptname,
2388                  char *username, int *status)
2389 1 {
2390 1     EDMRESubmitElement *se;
2391 1     int ret;

2393 1     if (status == NULL)
2394 2     {
2395 2         return -1;
2396 1     }

2398 1     if (scriptname == NULL || username == NULL)
2399 2     {
2400 2         *status = SUBMIT_BAD_PARAM;
2401 2         return -1;
2402 1     }

2404 1     ret = LookupSubmitElement(ID, elementID, &se, status);

2406 1     if (ret != 0)
2407 2     {
2408 2         return ret;
2409 1     }

2411 1     se -> setScriptName(scriptname);
2412 1     se -> setClientUserName(username);

2414 1     return 0;
2415  }
```

```
2417  /*****************************************************************************
                                                                          ********
2418  **
2419  ** Routine:  SetSEEbciConnect
2420  **
2421  ** Inputs:   int ID - submit ID
2422  **           int elementID - submit element ID
2423  **           char *hostname - the hostname to connect to
2424  **           int sockport - the port to connect to on the above host
2425  **
2426  ** Outputs:  int *status - a status of the operation
2427  **
2428  ** Return Codes:
2429  **           int - 0 for success and non-zero otherwise
2430  **
2431  ** Purpose:  Sets all ebci information.
2432  **
2433  *****************************************************************************
                                                                          ********
2434  */

2436  int
2437  SetSEEbciConnect(int ID, int elementID, char *hostname,
2438                   int sockport, int *status)
2439  1 {
2440  1     EDMRESubmitElement *se;
2441  1     int ret;

2443  1     if (status == NULL)
2444  2     {
2445  2         return -1;
2446  1     }

2448  1     if (hostname == NULL)
2449  2     {
2450  2         *status = SUBMIT_BAD_PARAM;
2451  2         return -1;
2452  1     }

2454  1     ret = LookupSubmitElement(ID, elementID, &se, status);

2456  1     if (ret != 0)
2457  2     {
2458  2         return ret;
2459  1     }

2461  1     se -> setSocketHost(hostname);
2462  1     se -> setSocketPort(sockport);

2464  1     return 0;
2465    }
```



```
2467  /*****************************************************************************
                                                                          ********
2468  **
2469  ** Routine:  SetSESummary
2470  **
2471  ** Inputs:   int ID - submit ID
2472  **           int elementID - submit element ID
2473  **           struct mark_summary *markptr - the marks for a given work
                                                                             item
2474  **
2475  ** Outputs:  int *status - a status of the operation
2476  **
2477  ** Return Codes:
2478  **           int - 0 for success and non-zero otherwise
2479  **
2480  ** Purpose:  Sets all submit element marks.
2481  **
2482  *****************************************************************************
                                                                          ********
2483  */

2485  int
2486  SetSESummary(int ID, int elementID, struct mark_summary *markptr,
2487               int *status)
2488  1 {
2489  1     EDMRESubmitElement *se;
2490  1     int ret;

2492  1     if (status == NULL)
2493  2     {
2494  2         return -1;
2495  1     }

2497  1     if (markptr == NULL)
2498  2     {
2499  2         *status = SUBMIT_BAD_PARAM;
2500  2         return -1;
2501  1     }

2503  1     ret = LookupSubmitElement(ID, elementID, &se, status);

2505  1     if (ret != 0)
2506  2     {
2507  2         return ret;
2508  1     }

2510  1     se -> setMarkSummary(markptr);

2512  1     return 0;
2513    }
```

```
2515  /*****************************************************************************
                                                                          ********
2516  **
2517  ** Routine:  SetSEVolumes
2518  **
2519  ** Inputs:   int ID - submit ID
2520  **           int elementID - submit element ID
2521  **           ebvl_volidlist_ty *vols - the volumes needed for this
2522  **                           part of restore
2523  **
2524  ** Outputs:  int *status - a status of the operation
2525  **
2526  ** Return Codes:
2527  **           int - 0 for success and non-zero otherwise
2528  **
2529  ** Purpose:  Sets the volumes needed to restore the submit element.
2530  **
2531  *****************************************************************************
                                                                          ********
2532  */

2534  int
2535  SetSEVolumes(int ID, int elementID, ebvl_volidlist_ty *vols,
2536               int *status)
2537 1 {
2538 1     EDMRESubmitElement *se;
2539 1     int ret;

2541 1     if (status == NULL)
2542 2     {
2543 2         return -1;
2544 1     }

2546 1     if (vols == NULL)
2547 2     {
2548 2         *status = SUBMIT_BAD_PARAM;
2549 2         return -1;
2550 1     }

2552 1     ret = LookupSubmitElement(ID, elementID, &se, status);

2554 1     if (ret != 0)
2555 2     {
2556 2         return ret;
2557 1     }

2559 1     se -> setVolumeList(vols);

2561 1     return 0;
2562  }
```



```
2564  /*****************************************************************************
                                                                          ********
2565  **
2566  ** Routine:  SetSESubmitFile
2567  **
2568  ** Inputs:   int ID - submit ID
2569  **           int elementID - submit element ID
2570  **
2571  ** Outputs:  int *status - a status of the operation
2572  **
2573  ** Return Codes:
2574  **           int - 0 for success and non-zero otherwise
2575  **
2576  ** Purpose:  Sets the submit file for the submit element.
2577  **
2578  *****************************************************************************
                                                                          ********
2579  */

2581  int
2582  SetSESubmitFile(int ID, int elementID, char *submitfile, int *status)
2583 1 {
2584 1     EDMRESubmitElement *se;
2585 1     int ret;

2587 1     if (status == NULL)
2588 2     {
2589 2         return -1;
2590 1     }

2592 1     if (submitfile == NULL)
2593 2     {
2594 2         *status = SUBMIT_BAD_PARAM;
2595 2         return -1;
2596 1     }

2598 1     ret = LookupSubmitElement(ID, elementID, &se, status);

2600 1     if (ret != 0)
2601 2     {
2602 2         return ret;
2603 1     }

2605 1     se -> setSubmitFile(submitfile, status);

2607 1     return 0;
2608  }
```

```
2610  /*************************************************************************
                                                                        ********
2611   **
2612   ** Routine:  SetSEPluginData
2613   **
2614   ** Inputs:   int ID - submit ID
2615   **           int elementID - submit element ID
2616   **           RWCollection *plugin_data - pointer to plugin specific
                                                        submit element data
2617   **
2618   ** Outputs:  int *status - a status of the operation
2619   **
2620   ** Return Codes:
2621   **           int - 0 for success and non-zero otherwise
2622   **
2623   ** Purpose:  Sets the pointer to the plugin specific data for the
                                                      submit element
2624   **
2625   *************************************************************************
                                                                        ********
2626   */

2628  int
2629  SetSEPluginData(
          int ID, int elementID, RWCollection *plugin_data, int *status)
2630  1 {
2631  1     EDMRESubmitElement *se;
2632  1     int ret;

2634  1     if (status == NULL)
2635  2     {
2636  2         return -1;
2637  1     }

2639  1     if (plugin_data == NULL)
2640  2     {
2641  2         *status = SUBMIT_BAD_PARAM;
2642  2         return -1;
2643  1     }

2645  1     ret = LookupSubmitElement(ID, elementID, &se, status);

2647  1     if (ret != 0)
2648  2     {
2649  2         return ret;
2650  1     }

2652  1     se -> setPluginData(plugin_data);

2654  1     return 0;
2655  }
```

```
/*
** Copyright 1996,1997 EMC Corporation
*/


/* EDMRESubmitElement.cc
 *
 *
 * Mission Statement:  file that contains the Handle class methods
 *
 * Primary Data Acted On:
 *
 * Compile-Time Options:
 *
 * Basic idea here:
 *                      The Handle object is a container which holds a
 *                      set of handles for each running auxproc.
 */

#if !defined(lint)
static char     RCS_id [] = "@(#)$RCSfile: EDMRESubmitElement.cc,v $ "
                            "$Revision: 1.0 $ "
                            "$Date: 1997/02/06 20:49:15 $" ;
#endif

#include <esl/c_portable.h>
#include <esl/ep_xopen.h>
#include <esl/inout.h>

#include <string.h>
#include <stdlib.h>

// Rogue Wave includes
#include <rw/collect.h>
#include <rw/rwfile.h>
#include <rw/vstream.h>
#include <rw/colclass.h>

#include <restore/RestoreObjectID.h>
#include <restore/dispatch_daemon.h>
#include <restore/restore_api.h>
#include <restore/RSLcontext.h>
#include <ebreport/ebvl.h>
#include <restore/EDMRESubmitElement.h>

// Needed for rougue wave linked list manager.
// 409 is the object ID.
RWDEFINE_COLLECTABLE(EDMRESubmitElement, EDMRESUBMITELEMENT)

/*************************************************************************
                                                                ********
**
** Routine:  EDMRESubmitElement constructor
**
** Inputs:   None
**
** Outputs:  None
**
** Return Codes:
**           None
**
** Purpose:  Initializes the Submit Element class by resetting the
**           internal data to 0.
**
*************************************************************************
                                                                ********
```



```
*/

EDMRESubmitElement::EDMRESubmitElement()
{
  submitElementID = 0;
  se_work_item = NULL;
  se_template = NULL;
  se_is_trailset_alternate = FALSE;
  se_client_rbuname = NULL;

  se_client_hostname = NULL;
  se_source_client_hostname  = NULL;
  se_is_inPlace = TRUE;
  se_dirtop = NULL;
  se_directory = NULL;
  se_policy = Older_Only_Overwrite;

  se_client_scriptname = NULL;
  se_socket_host = NULL;
  se_socket_port = 0;
  se_transport = restoreTransportNetwork;

  memset(&se_submit_summary, 0, sizeof(struct mark_summary));

  se_work_item_volume_list = NULL;
                            /* volumes needed for restore */
  se_submit_file_name = NULL;

  wi_type = 0;
}

/*************************************************************************
                                                                ********
**
** Routine:  EDMRESubmitElement destructor
**
** Inputs:   None
**
** Outputs:  None
**
** Return Codes:
**           None
**
** Purpose:  Doesn't really do anything but seems to be a requirement
**           for the linked list manager.
**
*************************************************************************
                                                                ********
*/
EDMRESubmitElement::~EDMRESubmitElement()
{
    if (se_work_item != NULL)
        free(se_work_item);

    if (se_template != NULL)
        free(se_template);

    if (se_client_rbuname != NULL)
        free(se_client_rbuname);

    if (se_client_hostname != NULL)
        free(se_client_hostname);

    if (se_source_client_hostname != NULL)
        free(se_source_client_hostname);
```

```
129  1      if (se_dirtop != NULL)
130  1          free(se_dirtop);

132  1      if (se_directory != NULL)
133  1          free(se_directory);

135  1      if (se_client_scriptname != NULL)
136  1          free(se_client_scriptname);

138  1      if (se_socket_host != NULL)
139  1          free(se_socket_host);

141  1      if (se_submit_file_name != NULL)
142  1          free(se_submit_file_name);

144  1      // Need to add some free routines for the mark_summary and
145  1      // the vollid stuff.
146  1      // We also have to make sure the submit file is unlinked before
147  1      // the submit file is free'd
148    }

150    /*****************************************************************************
                                                                              ********
151    **
152    ** Routine:  compareTo
153    **
154    ** Inputs:   RWCollectable *c - a pointer to the base class type which
155    **                              you can then cast and compare.
156    **
157    ** Outputs:  None
158    **
159    ** Return Codes:
160    **           int    -    returns numbers like qsort compare (-1, 0, 1)
161    **
162    ** Purpose:  Compare using the submit element ID.
163    **
164    *****************************************************************************
                                                                              ********
165    */

167    int
168    EDMRESubmitElement::compareTo(IN const RWCollectable *c) const
169  1 {
170  1      EDMRESubmitElement *localcmd = (EDMRESubmitElement *) c;

172  1      if (localcmd == NULL)
173  1          return -1;

175  1      if (localcmd -> submitElementID > submitElementID)
176  1          return 1;
177  1      else if (localcmd -> submitElementID < submitElementID)
178  1          return -1;

180  1      return 0;
181    }

183    /*****************************************************************************
                                                                              ********
184    **
185    ** Routine:  isEqual
186    **
187    ** Inputs:   RWCollectable *c - a pointer to the base class type which
188    **                              you can then cast and compare.
189    **
190    ** Outputs:  None
191    **
```



```
192    ** Return Codes:
193    **           RWBoolean    -    TRUE or FALSE
194    **
195    ** Purpose:  Compare pipes to find which work item needs service.
196    **
197    *****************************************************************************
                                                                              ********
198    */

200    RWBoolean
201    EDMRESubmitElement::isEqual(IN const RWCollectable *c) const
202  1 {
203  1      EDMRESubmitElement *localcmd = (EDMRESubmitElement *) c;

205  1      if (localcmd == NULL)
206  1          return FALSE;

208  1      if (localcmd -> submitElementID == submitElementID)
209  1          return TRUE;

211  1      return FALSE;
212    }

214    /*****************************************************************************
                                                                              ********
215    **
216    ** Routine:  hash
217    **
218    ** Inputs:   None
219    **
220    ** Outputs:  None
221    **
222    ** Return Codes:
223    **           unsigned    -    returns time started
224    **
225    ** Purpose:  Returns unique value, in this case submit element ID.
226    **
227    *****************************************************************************
                                                                              ********
228    */

230    unsigned
231    EDMRESubmitElement::hash() const
232  1 {
233  1      return (unsigned) submitElementID;
234    }

236    /*****************************************************************************
                                                                              ********
237    **
238    ** Routine:  saveGuts
239    **
240    ** Inputs:   RWFile  f  - File pointer where data will be saved.
241    **
242    ** Outputs:  None
243    **
244    ** Return Codes:
245    **           None
246    **
247    ** Purpose:  Save class internal data to a file.
248    **
249    *****************************************************************************
                                                                              ********
250    */

252    void
```

```
253   EDMRESubmitElement::saveGuts(IN RWFile &f)
254 1 {
255 1     // Save parent class data too
256 1     RWCollectable::saveGuts(f);
258 1     // Left as an exercise
259   }
261   /************************************************************************
                                                                        ********
262   **
263   ** Routine:  saveGuts
264   **
265   ** Inputs:   RWvostream strm - stream to write internal data to.
266   **
267   ** Outputs:  None
268   **
269   ** Return Codes:
270   **           None
271   **
272   ** Purpose:  Save class data to a stream.
273   **
274   ************************************************************************
                                                                        ********
275   */
277   void
278   EDMRESubmitElement::saveGuts(IN RWvostream &strm)
279 1 {
280 1     // Save parent class data too
281 1     RWCollectable::saveGuts(strm);
283 1     // Left as an exercise
284   }
286   /************************************************************************
                                                                        ********
287   **
288   ** Routine:  restoreGuts
289   **
290   ** Inputs:   RWFile f - file to read internal data from.
291   **
292   ** Outputs:  None
293   **
294   ** Return Codes:
295   **           None
296   **
297   ** Purpose:  Restores an instance of the Handle class by reading the
                                                                          data
298   **           from the passed in file.
299   **
300   ************************************************************************
                                                                        ********
301   */
303   void
304   EDMRESubmitElement::restoreGuts(IN RWFile &f)
305 1 {
306 1     // Restore parent data too
307 1     RWCollectable::restoreGuts(f);
309 1     // Left as an exercise
310   }
312   /************************************************************************
                                                                        ********
```



```
313   **
314   ** Routine:  restoreGuts
315   **
316   ** Inputs:   RWvistream strm - stream to read internal data from.
317   **
318   ** Outputs:  None
319   **
320   ** Return Codes:
321   **           None
322   **
323   ** Purpose:  Restores an instance of the Handle class by reading the
                                                                          data
324   **           from the passed in stream.
325   **
326   ************************************************************************
                                                                        ********
327   */
329   void
330   EDMRESubmitElement::restoreGuts(IN RWvistream &strm)
331 1 {
332 1     // Restore parent data too
333 1     RWCollectable::restoreGuts(strm);
335 1     // Left as an exercise
336   }
338   /************************************************************************
                                                                        ********
339   **
340   ** Routine:  binaryStoreSize
341   **
342   ** Inputs:   None
343   **
344   ** Outputs:  None
345   **
346   ** Return Codes:
347   **           RWspace count - file size of class written to disk in
                                                                          bytes
348   **
349   ** Purpose:  Returns the size of class if it were stored on disk.
350   **
351   ************************************************************************
                                                                        ********
352   */
354   RWspace
355   EDMRESubmitElement::binaryStoreSize() const
356 1 {
357 1     RWspace count = RWCollectable::binaryStoreSize() + 1;
359 1     // Left as an exercise
361 1     return count;
362   }
364   /************************************************************************
                                                                        ********
365   **
366   ** Routine:  getSubmitElementID
367   **
368   ** Inputs:   None
369   **
370   ** Outputs:  None
371   **
372   ** Return Codes:
```

```
373     **           The submit element ID
374     **
375     ** Purpose:  Returns the submit element ID.
                                        Will be 0 if not initialized fully.
376     **
377     ************************************************************************
                                                                        ********
378     */

380     int
381     EDMRESubmitElement::getSubmitElementID()
382 1   {
383 1       return submitElementID;
384     }


387     /************************************************************************
                                                                        ********
388     **
389     ** Routine:  setSubmitElementID
390     **
391     ** Inputs:   int se_id - the ID for the submit element.
                                                        Should be set before
392     **                       being inserted in the list.
393     **
394     ** Outputs:  None
395     **
396     ** Return Codes:
397     **           None
398     **
399     ** Purpose:  Sets the submit element ID.
400     **
401     ************************************************************************
                                                                        ********
402     */

404     void
405     EDMRESubmitElement::setSubmitElementID(int se_id)
406 1   {
407 1       submitElementID = se_id;
408     }

410     /************************************************************************
                                                                        ********
411     **
412     ** Routine:  getWorkItem
413     **
414     ** Inputs:   int maxsize - max size to copy into buffer
415     **
416     ** Outputs:  char *workitem - place to put workitem
417     **
418     ** Return Codes:
419     **           None
420     **
421     ** Purpose:  Returns the work item name in the specified buffer.
422     **
423     ************************************************************************
                                                                        ********
424     */

426     void
427     EDMRESubmitElement::getWorkItem(char *workitem, int maxsize)
428 1   {
429 1       if (workitem != NULL && se_work_item != NULL)
430 2       {
431 2           strncpy(workitem, se_work_item, maxsize - 1);
```



```
432 2           workitem[maxsize - 1] = 0;
433 1       }
434 1       else if (workitem != NULL)
435 2       {
436 2           workitem[0] = 0;
437 1       }
438     }

440     /************************************************************************
                                                                        ********
441     **
442     ** Routine:  setWorkItem
443     **
444     ** Inputs:   char *workitem - the workitem to copy
445     **
446     ** Outputs:  None
447     **
448     ** Return Codes:
449     **           None
450     **
451     ** Purpose:  Sets the work item name using the specified buffer.
452     **
453     ************************************************************************
                                                                        ********
454     */

456     void
457     EDMRESubmitElement::setWorkItem(char *workitem)
458 1   {
459 1       if (workitem != NULL)
460 2       {
461 2           if (se_work_item != NULL)
462 2               free(se_work_item);

464 2           se_work_item = strdup(workitem);
465 1       }
466     }

468     /************************************************************************
                                                                        ********
469     **
470     ** Routine:  getTemplate
471     **
472     ** Inputs:   int maxsize - max size to copy into buffer
473     **
474     ** Outputs:  char *template_name - place to put template name
475     **
476     ** Return Codes:
477     **           None
478     **
479     ** Purpose:  Returns the template name in the specified buffer.
480     **
481     ************************************************************************
                                                                        ********
482     */

484     void
485     EDMRESubmitElement::getTemplate(char *template_name, int maxsize)
486 1   {
487 1       if (template_name != NULL && se_template != NULL)
488 2       {
489 2           strncpy(template_name, se_template, maxsize - 1);
490 2           template_name[maxsize - 1] = 0;
491 1       }
492 1       else if (template_name != NULL)
493 2       {
```

```
494  2          template_name[0] = 0;
495  1      }
496     }

498     /*****************************************************************
                                                                      ********
499     **
500     ** Routine:  setTemplate
501     **
502     ** Inputs:   char *template_name - the template to copy
503     **
504     ** Outputs:  None
505     **
506     ** Return Codes:
507     **           None
508     **
509     ** Purpose:  Sets the template name using the specified buffer.
510     **
511     *****************************************************************
                                                                      ********
512     */

514     void
515     EDMRESubmitElement::setTemplate(char *template_name)
516  1  {
517  1      if (template_name != NULL)
518  2      {
519  2          if (se_template != NULL)
520  2              free(se_template);

522  2          se_template = strdup(template_name);
523  1      }
524     }

526     /*****************************************************************
                                                                      ********
527     **
528     ** Routine:  IsAlternateTrailset
529     **
530     ** Inputs:   None
531     **
532     ** Outputs:  None
533     **
534     ** Return Codes:
535     **           boolean value of se_is_trailset_alternate element
536     **
537     ** Purpose:  Returns the TRUE if alternate trailset used.
538     **
539     *****************************************************************
                                                                      ********
540     */

542     boolean_ty
543     EDMRESubmitElement::IsAlternateTrailset()
544  1  {
545  1      return se_is_trailset_alternate;
546     }

548     /*****************************************************************
                                                                      ********
549     **
550     ** Routine:  SetAlternateTrailset
551     **
552     ** Inputs:   boolean_ty - set it to TRUE or FALSE
553     **
554     ** Outputs:  None
```



```
555     **
556     ** Return Codes:
557     **           None
558     **
559     ** Purpose:  Sets the se_is_trailset_alternate variable to the given
                                                                      parameter
560     **
561     *****************************************************************
                                                                      ********
562     */

564     void
565     EDMRESubmitElement::SetAlternateTrailset(boolean_ty alt)
566  1  {
567  1      se_is_trailset_alternate = alt;
568     }

570     /*****************************************************************
                                                                      ********
571     **
572     ** Routine:  getClientUserName
573     **
574     ** Inputs:   int maxsize - max size to copy into buffer
575     **
576     ** Outputs:  char *clientname - place to put user name to use on
                                                                      client
577     **
578     ** Return Codes:
579     **           None
580     **
581     ** Purpose:  Returns the user name in the specified buffer.
582     **
583     *****************************************************************
                                                                      ********
584     */

586     void
587     EDMRESubmitElement::getClientUserName(char *clientuname, int maxsize)
588  1  {
589  1      if (clientuname != NULL && se_client_rbuname != NULL)
590  2      {
591  2          strncpy(clientuname, se_client_rbuname, maxsize - 1);
592  2          clientuname[maxsize - 1] = 0;
593  1      }
594  1      else if (clientuname != NULL)
595  2      {
596  2          clientuname[0] = 0;
597  1      }
598     }

600     /*****************************************************************
                                                                      ********
601     **
602     ** Routine:  setClientUserName
603     **
604     ** Inputs:   char *clientuname - the user name to use on client
605     **
606     ** Outputs:  None
607     **
608     ** Return Codes:
609     **           None
610     **
611     ** Purpose:  Sets the user name to use on the client.
612     **
613     *****************************************************************
                                                                      ********
```

```
614     */
616     void
617     EDMRESubmitElement::setClientUserName(char *clientuname)
618   1 {
619   1     if (clientuname != NULL)
620   2     {
621   2         if (se_client_rbuname != NULL)
622   2             free(se_client_rbuname);
624   2         se_client_rbuname = strdup(clientuname);
625   1     }
626     }
628     /************************************************************************
                                                                        ********
629     **
630     ** Routine:  getClientSource
631     **
632     ** Inputs:   int maxsize - max size to copy into buffer
633     **
634     ** Outputs:  char *clientname - place to put client that was backed up
635     **
636     ** Return Codes:
637     **           None
638     **
639     ** Purpose:  Returns the client name in the specified buffer.
640     **
641     ************************************************************************
                                                                        ********
642     */
644     void
645     EDMRESubmitElement::getClientSource(char *clientname, int maxsize)
646   1 {
647   1     if (clientname != NULL && se_source_client_hostname != NULL)
648   2     {
649   2         strncpy(clientname, se_source_client_hostname, maxsize - 1);
650   2         clientname[maxsize - 1] = 0;
651   1     }
652   1     else if (clientname != NULL)
653   2     {
654   2         clientname[0] = 0;
655   1     }
656     }
658     /************************************************************************
                                                                        ********
659     **
660     ** Routine:  setClientSource
661     **
662     ** Inputs:   char *clientname - the client that was backed up
663     **
664     ** Outputs:  None
665     **
666     ** Return Codes:
667     **           None
668     **
669     ** Purpose:  Sets the client name that was backed up using the
670     **           specified buffer.
671     **
672     ************************************************************************
                                                                        ********
673     */
675     void
```



```
676     EDMRESubmitElement::setClientSource(char *clientname)
677   1 {
678   1     if (clientname != NULL)
679   2     {
680   2         if (se_source_client_hostname != NULL)
681   2             free(se_source_client_hostname);
683   2         se_source_client_hostname = strdup(clientname);
684   1     }
685     }
687     /************************************************************************
                                                                        ********
688     **
689     ** Routine:  getClientHostname
690     **
691     ** Inputs:   int maxsize - max size to copy into buffer
692     **
693     ** Outputs:  char *clienthost - place to put the client hostname
694     **
695     ** Return Codes:
696     **           None
697     **
698     ** Purpose:  Returns the client host name in the specified buffer.
699     **
700     ************************************************************************
                                                                        ********
701     */
703     void
704     EDMRESubmitElement::getClientHostname(char *clienthost, int maxsize)
705   1 {
706   1     if (clienthost != NULL && se_client_hostname != NULL)
707   2     {
708   2         strncpy(clienthost, se_client_hostname, maxsize - 1);
709   2         clienthost[maxsize - 1] = 0;
710   1     }
711   1     else if (clienthost != NULL)
712   2     {
713   2         clienthost[0] = 0;
714   1     }
715     }
717     /************************************************************************
                                                                        ********
718     **
719     ** Routine:  setClientHostname
720     **
721     ** Inputs:   char *clienthost - the client hostname
722     **
723     ** Outputs:  None
724     **
725     ** Return Codes:
726     **           None
727     **
728     ** Purpose:  Sets the client host name using the specified buffer.
729     **
730     ************************************************************************
                                                                        ********
731     */
733     void
734     EDMRESubmitElement::setClientHostname(char *clienthost)
735   1 {
736   1     if (clienthost != NULL)
737   2     {
```

```
738 2          if (se_client_hostname != NULL)
739 2              free(se_client_hostname);

741 2          se_client_hostname = strdup(clienthost);
742 1      }
743    }

745    /*************************************************************************
                                                                        ********
746    **
747    ** Routine:  IsInPlace()
748    **
749    ** Inputs:   None
750    **
751    ** Outputs:  None
752    **
753    ** Return Codes:
754    **           boolean - is_inPlace variable
755    **
756    ** Purpose:  Returns TRUE if work item is to be restored to the same
                                                                          place
757    **
758    *************************************************************************
                                                                        ********
759    */

761    boolean_ty
762    EDMRESubmitElement::IsInPlace()
763 1  {
764 1      return se_is_inPlace;
765    }

767    /*************************************************************************
                                                                        ********
768    **
769    ** Routine:  setInPlace
770    **
771    ** Inputs:   boolean_ty inplace - a boolean to set whether a restore
                                                                             is
772    **                       in-place or not.
773    **
774    ** Outputs:  None
775    **
776    ** Return Codes:
777    **           None
778    **
779    ** Purpose:  Sets the se_is_inPlace variable.
780    **
781    *************************************************************************
                                                                        ********
782    */

784    void
785    EDMRESubmitElement::setInPlace(boolean_ty inplace)
786 1  {
787 1      se_is_inPlace = inplace;
788    }

790    /*************************************************************************
                                                                        ********
791    **
792    ** Routine:  getDirectoryTop
793    **
794    ** Inputs:   int maxsize - max size to copy into buffer
795    **
796    ** Outputs:  char *dirtop - place to put head of directory
```



```
797    **
798    ** Return Codes:
799    **           None
800    **
801    ** Purpose:  Returns the directory top in the specified buffer
802    **
803    *************************************************************************
                                                                        ********
804    */

806    void
807    EDMRESubmitElement::getDirectoryTop(char *dirtop, int maxsize)
808 1  {
809 1      if (dirtop != NULL && se_dirtop != NULL)
810 2      {
811 2          strncpy(dirtop, se_dirtop, maxsize - 1);
812 2          dirtop[maxsize - 1] = 0;
813 1      }
814 1      else if (dirtop != NULL)
815 2      {
816 2          dirtop[0] = 0;
817 1      }
818    }

820    /*************************************************************************
                                                                        ********
821    **
822    ** Routine:  setDirectoryTop
823    **
824    ** Inputs:   char *dirtop - the head of directory
825    **
826    ** Outputs:  None
827    **
828    ** Return Codes:
829    **           None
830    **
831    ** Purpose:  Sets the directory top using the specified buffer.
832    **
833    *************************************************************************
                                                                        ********
834    */

836    void
837    EDMRESubmitElement::setDirectoryTop(char *dirtop)
838 1  {
839 1      if (dirtop != NULL)
840 2      {
841 2          if (se_dirtop != NULL)
842 2              free(se_dirtop);

844 2          se_dirtop = strdup(dirtop);
845 1      }
846    }

848    /*************************************************************************
                                                                        ********
849    **
850    ** Routine:  getDirectoryDestination
851    **
852    ** Inputs:   int maxsize - max size to copy into buffer
853    **
854    ** Outputs:  char *directory - place to put the restore
855    **
856    ** Return Codes:
857    **           None
858    **
```

```
859     ** Purpose:  Returns the directory in the specified buffer.
860     **
861     ************************************************************************
                                                                        ********
862     */

864     void
865     EDMRESubmitElement::getDirectoryDestination(
                                                char *directory, int maxsize)
866   1 {
867   1     if (directory != NULL && se_directory != NULL)
868   2     {
869   2         strncpy(directory, se_directory, maxsize - 1);
870   2         directory[maxsize - 1] = 0;
871   1     }
872   1     else if (directory != NULL)
873   2     {
874   2         directory[0] = 0;
875   1     }
876     }

878     /************************************************************************
                                                                        ********
879     **
880     ** Routine:  setDirectoryDestination
881     **
882     ** Inputs:   char *directory - place to put the restore
883     **
884     ** Outputs:  None
885     **
886     ** Return Codes:
887     **           None
888     **
889     ** Purpose:  Sets the destination for the restore using the specified
                                                                        buffer.
890     **
891     ************************************************************************
                                                                        ********
892     */

894     void
895     EDMRESubmitElement::setDirectoryDestination(char *directory)
896   1 {
897   1     if (directory != NULL)
898   2     {
899   2         if (se_directory != NULL)
900   2             free(se_directory);

902   2         se_directory = strdup(directory);
903   1     }
904     }

906     /************************************************************************
                                                                        ********
907     **
908     ** Routine:  getRestoreTransport
909     **
910     ** Inputs:   None
911     **
912     ** Outputs:  None
913     **
914     ** Return Codes:
915     **           RestoreTransport - as enumerated in restore_api.h
916     **
917     ** Purpose:  Returns the restore transport
918     **
```



```
919     ************************************************************************
                                                                        ********
920     */

922     RestoreTransport
923     EDMRESubmitElement::getRestoreTransport()
924   1 {
925   1     return se_transport;
926     }

928     /************************************************************************
                                                                        ********
929     **
930     ** Routine:  setRestoreTransport
931     **
932     ** Inputs:   RestoreTransport transpt - use this to set the restore
                                                                        transport
933     **
934     ** Outputs:  None
935     **
936     ** Return Codes:
937     **           None
938     **
939     ** Purpose:  Sets the restore transport for the restore using the
940     **           specified parameter.
941     **
942     ************************************************************************
                                                                        ********
943     */

945     void
946     EDMRESubmitElement::setRestoreTransport(RestoreTransport transpt)
947   1 {
948   1     se_transport = transpt;
949     }

951     /************************************************************************
                                                                        ********
952     **
953     ** Routine:  getOverwritePolicy
954     **
955     ** Inputs:   None
956     **
957     ** Outputs:  None
958     **
959     ** Return Codes:
960     **           OverwritePolicy - as enumerated in restore_api.h
961     **
962     ** Purpose:  Returns the overwrite policy
963     **
964     ************************************************************************
                                                                        ********
965     */

967     OverwritePolicy
968     EDMRESubmitElement::getOverwritePolicy()
969   1 {
970   1     return se_policy;
971     }

973     /************************************************************************
                                                                        ********
974     **
975     ** Routine:  setOverwritePolicy
976     **
977     ** Inputs:   OverwritePolicy policy - use this to set the overwrite
```

```
                                                                    policy
 978    **
 979    ** Outputs:  None
 980    **
 981    ** Return Codes:
 982    **           None
 983    **
 984    ** Purpose:  Sets the overwrite policy for the restore using the
 985    **           specified parameter.
 986    **
 987    ************************************************************************
                                                                        ********
 988    */

 990    void
 991    EDMRESubmitElement::setOverwritePolicy(OverwritePolicy policy)
 992 1  {
 993 1      se_policy = policy;
 994    }

 996    /************************************************************************
                                                                        ********
 997    **
 998    ** Routine:  getScriptName
 999    **
1000    ** Inputs:   int maxsize - max size to copy into buffer
1001    **
1002    ** Outputs:  char *scriptname - place to put the script that will run
                                                                      on client
1003    **
1004    ** Return Codes:
1005    **           None
1006    **
1007    ** Purpose:  Returns the script name in the specified buffer.
1008    **
1009    ************************************************************************
                                                                        ********
1010    */

1012    void
1013    EDMRESubmitElement::getScriptName(char *scriptname, int maxlen)
1014 1  {
1015 1      if (scriptname != NULL && se_client_scriptname != NULL)
1016 2      {
1017 2          strncpy(scriptname, se_client_scriptname, maxlen - 1);
1018 2          scriptname[maxlen - 1] = 0;
1019 1      }
1020 1      else if (scriptname != NULL)
1021 2      {
1022 2          scriptname[0] = 0;
1023 1      }
1024   }

1026   /************************************************************************
                                                                        ********
1027   **
1028   ** Routine:  setScriptName
1029   **
1030   ** Inputs:   char *scriptname - script to run on the client
1031   **
1032   ** Outputs:  None
1033   **
1034   ** Return Codes:
1035   **           None
1036   **
1037   ** Purpose:  Sets the scriptname to be used on the client using the
```



```
1038   **           specified buffer.
1039   **
1040   ************************************************************************
                                                                        ********
1041   */

1043   void
1044   EDMRESubmitElement::setScriptName(char *scriptname)
1045 1 {
1046 1     if (scriptname != NULL)
1047 2     {
1048 2         if (se_client_scriptname != NULL)
1049 2             free(se_client_scriptname);

1051 2         se_client_scriptname = strdup(scriptname);
1052 1     }
1053   }

1055   /************************************************************************
                                                                        ********
1056   **
1057   ** Routine:  getSocketHost
1058   **
1059   ** Inputs:   int maxsize - max size to copy into buffer
1060   **
1061   ** Outputs:  char *sockethost - place to put hostname to connect to
1062   **
1063   ** Return Codes:
1064   **           None
1065   **
1066   ** Purpose:  Returns the hostname to use in the specified buffer.
1067   **
1068   ************************************************************************
                                                                        ********
1069   */

1071   void
1072   EDMRESubmitElement::getSocketHost(char *sockethost, int maxsize)
1073 1 {
1074 1     if (sockethost != NULL && se_socket_host != NULL)
1075 2     {
1076 2         strncpy(sockethost, se_socket_host, maxsize - 1);
1077 2         sockethost[maxsize - 1] = 0;
1078 1     }
1079 1     else if (sockethost != NULL)
1080 2     {
1081 2         sockethost[0] = 0;
1082 1     }
1083   }

1085   /************************************************************************
                                                                        ********
1086   **
1087   ** Routine:  setSocketHost
1088   **
1089   ** Inputs:   char *sockethost - hostname to connect to
1090   **
1091   ** Outputs:  None
1092   **
1093   ** Return Codes:
1094   **           None
1095   **
1096   ** Purpose:  Sets the hostname to connect to using the specified
                                                                      buffer.
1097   **
1098   ************************************************************************
```

```
                                                                        ********
1099    */

1101    void
1102    EDMRESubmitElement::setSocketHost(char *sockethost)
1103  1 {
1104  1     if (sockethost != NULL)
1105  2     {
1106  2         if (se_socket_host != NULL)
1107  2             free(se_socket_host);

1109  2         se_socket_host = strdup(sockethost);
1110  1     }
1111    }

1113    /*************************************************************************
                                                                        ********
1114    **
1115    ** Routine:  getSocketPort
1116    **
1117    ** Inputs:   None
1118    **
1119    ** Outputs:  None
1120    **
1121    ** Return Codes:
1122    **           int - the port to connect to
1123    **
1124    ** Purpose:  Returns the port to connect to.
1125    **
1126    *************************************************************************
                                                                        ********
1127    */

1129    int
1130    EDMRESubmitElement::getSocketPort()
1131  1 {
1132  1     return se_socket_port;
1133    }

1135    /*************************************************************************
                                                                        ********
1136    **
1137    ** Routine:  setSocketPort
1138    **
1139    ** Inputs:   int socketport - port to connect to
1140    **
1141    ** Outputs:  None
1142    **
1143    ** Return Codes:
1144    **           None
1145    **
1146    ** Purpose:  Sets the destination port for the restore using the
1147    **           specified parameter.
1148    **
1149    *************************************************************************
                                                                        ********
1150    */

1152    void
1153    EDMRESubmitElement::setSocketPort(int socketport)
1154  1 {
1155  1     se_socket_port = socketport;
1156    }

1158    /*************************************************************************
                                                                        ********
```



```
1159    **
1160    ** Routine:  getMarkSummary
1161    **
1162    ** Inputs:   None
1163    **
1164    ** Outputs:  struct mark_summary *markptr - place to put mark summary
1165    **
1166    ** Return Codes:
1167    **           None
1168    **
1169    ** Purpose:  Returns the mark summary in the specified buffer.
1170    **
1171    *************************************************************************
                                                                        ********
1172    */

1174    void
1175    EDMRESubmitElement::getMarkSummary(struct mark_summary *markptr)
1176  1 {
1177  1     if (markptr != NULL)
1178  1         memcpy(markptr, &se_submit_summary, sizeof(
                                                    struct mark_summary));
1179    }

1181    /*************************************************************************
                                                                        ********
1182    **
1183    ** Routine:  setMarkSummary
1184    **
1185    ** Inputs:   struct mark_summary *markptr - place to put the mark
                                                                        summary
1186    **
1187    ** Outputs:  None
1188    **
1189    ** Return Codes:
1190    **           None
1191    **
1192    ** Purpose:  Sets the mark summary using the specified buffer.
1193    **
1194    *************************************************************************
                                                                        ********
1195    */

1197    void
1198    EDMRESubmitElement::setMarkSummary(struct mark_summary *markptr)
1199  1 {
1200  1     if (markptr != NULL)
1201  1         memcpy(&se_submit_summary, markptr, sizeof(
                                                    struct mark_summary));
1202    }

1204    /*************************************************************************
                                                                        ********
1205    **
1206    ** Routine:  getVolumeList
1207    **
1208    ** Inputs:   None
1209    **
1210    ** Outputs:  ebvl_volidlist_ty **volptr - place to put the volume list
1211    **
1212    ** Return Codes:
1213    **           None
1214    **
1215    ** Purpose:  Returns the volume list in the specified buffer.
1216    **
1217    *************************************************************************
```

```
                                                                        ********
 1218   */

 1220   void
 1221   EDMRESubmitElement::getVolumeList(ebvl_volidlist_ty **volptr)
 1222 1 {
 1223 1     if (volptr != NULL)
 1224 1         *volptr = se_work_item_volume_list;
 1225   }

 1227   /*************************************************************************
                                                                        ********
 1228   **
 1229   ** Routine:  setVolumeList
 1230   **
 1231   ** Inputs:   ebvl_volidlist_ty *volptr - place to put the volume list
 1232   **
 1233   ** Outputs:  None
 1234   **
 1235   ** Return Codes:
 1236   **           None
 1237   **
 1238   ** Purpose:  Sets the volume list using the specified buffer.
 1239   **
 1240   *************************************************************************
                                                                        ********
 1241   */

 1243   void
 1244   EDMRESubmitElement::setVolumeList(ebvl_volidlist_ty *volptr)
 1245 1 {
 1246 1     if (volptr != NULL)
 1247 1         se_work_item_volume_list = volptr;
 1248   }

 1250   /*************************************************************************
                                                                        ********
 1251   **
 1252   ** Routine:  getSubmitFile
 1253   **
 1254   ** Inputs:   int maxlen - the size of the buffer passed in
 1255   **
 1256   ** Outputs:  char *submitfile - place to put the submit file name
 1257   **
 1258   ** Return Codes:
 1259   **           None
 1260   **
 1261   ** Purpose:  Returns the submit file name in the specified buffer.
 1262   **
 1263   *************************************************************************
                                                                        ********
 1264   */

 1266   void
 1267   EDMRESubmitElement::getSubmitFile(char *submitfile, int maxlen)
 1268 1 {
 1269 1     if (submitfile != NULL && se_submit_file_name != NULL)
 1270 2     {
 1271 2         strncpy(submitfile, se_submit_file_name, maxlen);
 1272 2         submitfile[maxlen - 1] = 0;
 1273 1     }
 1274 1     else if (submitfile != NULL)
 1275 2     {
 1276 2         submitfile[0] = 0;
 1277 1     }
 1278   }
```



```
 1280   /*************************************************************************
                                                                        ********
 1281   **
 1282   ** Routine:  setSubmitFile
 1283   **
 1284   ** Inputs:   None
 1285   **
 1286   ** Outputs:  int *status - return the status of this operation,
                                                                        generally
 1287   **                         the errno.
 1288   **
 1289   ** Return Codes:
 1290   **           None
 1291   **
 1292   ** Purpose:  Sets the submit file name and makes sure it is unique.
 1293   **
 1294   *************************************************************************
                                                                        ********
 1295   */

 1297   void
 1298   EDMRESubmitElement::setSubmitFile(char *submitfile, int *status)
 1299 1 {

 1301 1   if(NULL != submitfile)
 1302 2   {
 1303 2     if(NULL != se_submit_file_name)
 1304 3     {
 1305 3       free(se_submit_file_name);
 1306 2     }
 1307 2     se_submit_file_name = strdup(submitfile);
 1308 2     if(NULL == se_submit_file_name)
 1309 3     {
 1310 3       *status = ENOMEM;
 1311 2     }
 1312 1   }
 1313 1   else
 1314 2   {
 1315 2     *status = EINVAL;
 1316 1   }
 1317 1   return;
 1318   }

 1320   /*************************************************************************
                                                                        ********
 1321   **
 1322   ** Routine:  getSocketPort
 1323   **
 1324   ** Inputs:   None
 1325   **
 1326   ** Outputs:  None
 1327   **
 1328   ** Return Codes:
 1329   **           char - the work item type
 1330   **
 1331   ** Purpose:  Returns the work item type
 1332   **
 1333   *************************************************************************
                                                                        ********
 1334   */

 1336   char
 1337   EDMRESubmitElement::getWorkItemType()
 1338 1 {
 1339 1     return wi_type;
```

```
1340    }
1342    /*****************************************************************************
                                                                            ********
1343    **
1344    ** Routine:  setWorkItemType
1345    **
1346    ** Inputs:   char type - the work item type
1347    **
1348    ** Outputs:  None
1349    **
1350    ** Return Codes:
1351    **           None
1352    **
1353    ** Purpose:  Sets the work item type using the specified parameter.
1354    **
1355    *****************************************************************************
                                                                            ********
1356    */

1358    void
1359    EDMRESubmitElement::setWorkItemType(char type)
1360  1 {
1361  1     wi_type = type;
1362    }

1364    /*****************************************************************************
                                                                            ********
1365    **
1366    ** Routine:  getPluginData
1367    **
1368    ** Inputs:   none
1369    **
1370    ** Outputs:  RWCollection *pi_data: pointer to plug-in specific data
                                                                              object
1371    **
1372    ** Return Codes:
1373    **           None
1374    **
1375    ** Purpose:  Retrieves the pointer to the plugin specific submit
                                                                            element data
1376    **
1377    *****************************************************************************
                                                                            ********
1378    */

1380    RWCollection  *
1381    EDMRESubmitElement::getPluginData()
1382  1 {
1383  1     return plugin_data;
1384    }

1386    /*****************************************************************************
                                                                            ********
1387    **
1388    ** Routine:  setPluginData
1389    **
1390    ** Inputs:   RWCollection *pi_data: pointer to plug-in specific data
                                                                              object
1391    **
1392    ** Outputs:  None
1393    **
1394    ** Return Codes:
1395    **           None
1396    **
1397    ** Purpose:  Sets the pointer to the plugin specific submit element
```



```
                                                                              data
1398    **
1399    *****************************************************************************
                                                                            ********
1400    */

1402    void
1403    EDMRESubmitElement::setPluginData(RWCollection *pi_data)
1404  1 {
1405  1     plugin_data = pi_data;
1406    }
```

```
/*
** Copyright 1996,1997 EMC Corporation
*/


/* EDMRESubmitObj.cc
 *
 *
 * Mission Statement:  file that contains the Handle class methods
 *
 * Primary Data Acted On:
 *
 * Compile-Time Options:
 *
 * Basic idea here:
 *                     The Handle object is a container which holds a
 *                     set of handles for each running auxproc.
 */

#if !defined(lint)
static char     RCS_id [] = "@(#)$RCSfile: EDMRESubmitObj.cc,v $ "
                            "$Revision: 1.0 $ "
                            "$Date: 1997/02/06 20:49:15 $" ;
#endif

#include <esl/c_portable.h>
#include <esl/ep_xopen.h>
#include <esl/inout.h>

#include <string.h>
#include <stdlib.h>

// Rogue Wave includes
#include <rw/collect.h>
#include <rw/rwfile.h>
#include <rw/vstream.h>
#include <rw/bintree.h>

#include <restore/RestoreObjectID.h>
#include <restore/dispatch_daemon.h>
#include <restore/restore_api.h>
#include <restore/RSLcontext.h>
#include <ebreport/ebvl.h>

#include <restore/EDMRESubmitElement.h>
#include <restore/EDMRESubmitObj.h>

// Needed for rougue wave linked list manager.
// 408 is the object ID.
RWDEFINE_COLLECTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)

/*************************************************************************
                                                                ********
**
** Routine:  EDMRESubmitObj constructor
**
** Inputs:   None
**
** Outputs:  None
**
** Return Codes:
**           None
**
** Purpose:  Initializes the Submit Object class by resetting the
                                                                internal
**           data to 0.
```



```
**
*************************************************************************
                                                                ********
*/

EDMRESubmitObj::EDMRESubmitObj()
{
    submitID = 0;
    so_submit_type = 0;
    so_execute_flags = 0;
    so_no_vm_check = FALSE;

    pre_phase_executable = NULL;
    pre_phase_argv = NULL;
    pre_phase_env = NULL;

    execute_override_phase_executable = NULL;
    execute_override_phase_argv = NULL;
    execute_override_phase_env = NULL;

    post_phase_executable = NULL;
    post_phase_argv = NULL;
    post_phase_env = NULL;

    so_backup_administrator = FALSE;
    so_src_sys_admin = FALSE;
    so_dst_sys_admin = FALSE;

    so_human_uid = -1;
    so_human_uidname = NULL;
    so_effective_uid = -1;
    so_effective_uidname = NULL;

    memset(&so_total_submit_summary, 0, sizeof(struct mark_summary));
    so_total_volume_list = NULL;
    so_witem_count = 0;
}

/*************************************************************************
                                                                ********
**
** Routine:  EDMRESubmitObj destructor
**
** Inputs:   None
**
** Outputs:  None
**
** Return Codes:
**           None
**
** Purpose:  Doesn't really do anything but seems to be a requirement
**           for the linked list manager.
**
*************************************************************************
                                                                ********
*/
EDMRESubmitObj::~EDMRESubmitObj()
{
    submitElements.clearAndDestroy();

    if (pre_phase_executable != NULL)
        free(pre_phase_executable);

    if (execute_override_phase_executable != NULL)
        free(execute_override_phase_executable);
```

```
128 1     if (post_phase_executable != NULL)
129 1         free(post_phase_executable);

131 1     if (so_human_uidname != NULL)
132 1         free(so_human_uidname);

134 1     if (so_effective_uidname != NULL)
135 1         free(so_effective_uidname);

137 1 /*

139 1     Currently we aren't managing/creating this memory so don't free it.

141 1     if (pre_phase_argv != NULL)
142 1     {
143 1         int i = 0;
144 1         while(pre_phase_argv[i] != NULL)
145 1             free(pre_phase_argv[i++]);

147 1         free(pre_phase_argv);
148 1     }

150 1     if (pre_phase_env != NULL)
151 1     {
152 1         int i = 0;
153 1         while(pre_phase_env[i] != NULL)
154 1             free(pre_phase_env[i++]);

156 1         free(pre_phase_env);
157 1     }

159 1     if (execute_override_phase_argv != NULL)
160 1     {
161 1         int i = 0;
162 1         while(execute_override_phase_argv[i] != NULL)
163 1             free(execute_override_phase_argv[i++]);

165 1         free(execute_override_phase_argv);
166 1     }

168 1     if (execute_override_phase_env != NULL)
169 1     {
170 1         int i = 0;
171 1         while(execute_override_phase_env[i] != NULL)
172 1             free(execute_override_phase_env[i++]);

174 1         free(execute_override_phase_env);
175 1     }

177 1     if (post_phase_argv != NULL)
178 1     {
179 1         int i = 0;
180 1         while(post_phase_argv[i] != NULL)
181 1             free(post_phase_argv[i++]);

183 1         free(post_phase_argv);
184 1     }

186 1     if (post_phase_env != NULL)
187 1     {
188 1         int i = 0;
189 1         while(post_phase_env[i] != NULL)
190 1             free(post_phase_env[i++]);

192 1         free(post_phase_env);
193 1     }
```



```
194 1 */

196 1     // Need to free mark summary and volume summary here.
197   }

199   /*****************************************************************************
                                                                          ********

200   **
201   ** Routine:  compareTo
202   **
203   ** Inputs:   RWCollectable *c - a pointer to the base class type which
204   **                              you can then cast and compare.
205   **
206   ** Outputs:  None
207   **
208   ** Return Codes:
209   **           int    -    returns numbers like qsort compare (-1, 0, 1)
210   **
211   ** Purpose:  Compare using the submit ID.
212   **
213   *****************************************************************************
                                                                          ********

214   */

216   int
217   EDMRESubmitObj::compareTo(IN const RWCollectable *c) const
218 1 {
219 1     EDMRESubmitObj *localcmd = (EDMRESubmitObj *) c;

221 1     if (localcmd == NULL)
222 1         return -1;

224 1     if (localcmd -> submitID > submitID)
225 1         return 1;
226 1     else if (localcmd -> submitID < submitID)
227 1         return -1;

229 1     return 0;
230   }

232   /*****************************************************************************
                                                                          ********

233   **
234   ** Routine:  isEqual
235   **
236   ** Inputs:   RWCollectable *c - a pointer to the base class type which
237   **                              you can then cast and compare.
238   **
239   ** Outputs:  None
240   **
241   ** Return Codes:
242   **           RWBoolean    -    TRUE or FALSE
243   **
244   ** Purpose:  Compare submit IDs to see if it is the same submit object.
245   **
246   *****************************************************************************
                                                                          ********

247   */

249   RWBoolean
250   EDMRESubmitObj::isEqual(IN const RWCollectable *c) const
251 1 {
252 1     EDMRESubmitObj *localcmd = (EDMRESubmitObj *) c;

254 1     if (localcmd == NULL)
255 1         return FALSE;
```

```
257 1     if (localcmd -> submitID == submitID)
258 1         return TRUE;

260 1     return FALSE;
261   }

263   /*************************************************************************
                                                                        ********
264   **
265   ** Routine:  hash
266   **
267   ** Inputs:   None
268   **
269   ** Outputs:  None
270   **
271   ** Return Codes:
272   **           unsigned   -    returns submit ID.
273   **
274   ** Purpose:  Returns unique value, in this case submit ID.
275   **
276   *************************************************************************
                                                                        ********
277   */

279   unsigned
280   EDMRESubmitObj::hash() const
281 1 {
282 1     return (unsigned) submitID;
283   }

285   /*************************************************************************
                                                                        ********
286   **
287   ** Routine:  saveGuts
288   **
289   ** Inputs:   RWFile  f  - File pointer where data will be saved.
290   **
291   ** Outputs:  None
292   **
293   ** Return Codes:
294   **           None
295   **
296   ** Purpose:  Save class internal data to a file.
297   **
298   *************************************************************************
                                                                        ********
299   */

301   void
302   EDMRESubmitObj::saveGuts(IN RWFile &f)
303 1 {
304 1     // Save parent class data too
305 1     RWCollectable::saveGuts(f);

307 1     // Left as an exercise
308   }

310   /*************************************************************************
                                                                        ********
311   **
312   ** Routine:  saveGuts
313   **
314   ** Inputs:   RWvostream strm - stream to write internal data to.
315   **
316   ** Outputs:  None
```



```
317   **
318   ** Return Codes:
319   **           None
320   **
321   ** Purpose:  Save class data to a stream.
322   **
323   *************************************************************************
                                                                        ********
324   */

326   void
327   EDMRESubmitObj::saveGuts(IN RWvostream &strm)
328 1 {
329 1     // Save parent class data too
330 1     RWCollectable::saveGuts(strm);

332 1     // Left as an exercise
333   }

335   /*************************************************************************
                                                                        ********
336   **
337   ** Routine:  restoreGuts
338   **
339   ** Inputs:   RWFile f - file to read internal data from.
340   **
341   ** Outputs:  None
342   **
343   ** Return Codes:
344   **           None
345   **
346   ** Purpose:  Restores an instance of the Handle class by reading the
                                                                          data
347   **           from the passed in file.
348   **
349   *************************************************************************
                                                                        ********
350   */

352   void
353   EDMRESubmitObj::restoreGuts(IN RWFile &f)
354 1 {
355 1     // Restore parent data too
356 1     RWCollectable::restoreGuts(f);

358 1     // Left as an exercise
359   }

361   /*************************************************************************
                                                                        ********
362   **
363   ** Routine:  restoreGuts
364   **
365   ** Inputs:   RWvistream strm - stream to read internal data from.
366   **
367   ** Outputs:  None
368   **
369   ** Return Codes:
370   **           None
371   **
372   ** Purpose:  Restores an instance of the Handle class by reading the
                                                                          data
373   **           from the passed in stream.
374   **
375   *************************************************************************
                                                                        ********
```

```
376     */
378     void
379     EDMRESubmitObj::restoreGuts(IN RWvistream &strm)
380   1 {
381   1     // Restore parent data too
382   1     RWCollectable::restoreGuts(strm);
384   1     // Left as an exercise
385     }
387     /************************************************************************
                                                                        ********
388     **
389     ** Routine:  binaryStoreSize
390     **
391     ** Inputs:   None
392     **
393     ** Outputs:  None
394     **
395     ** Return Codes:
396     **           RWspace count - file size of class written to disk in
                                                                          bytes
397     **
398     ** Purpose:  Returns the size of class if it were stored on disk.
399     **
400     ************************************************************************
                                                                        ********
401     */
403     RWspace
404     EDMRESubmitObj::binaryStoreSize() const
405   1 {
406   1     RWspace count = RWCollectable::binaryStoreSize() +
407   1                     sizeof(submitID);
408   1     // Left as an exercise
409   1     return count;
410     }
411     /************************************************************************
                                                                        ********
412     **
413     ** Routine: deleteSubmitFiles
414     **
415     ** Inputs:   None
416     **
417     ** Outputs:  None
418     **
419     ** Return Codes:
420     **           int - the submit ID
421     **
422     ** Purpose:  Returns the submit ID.
423     **
424     ************************************************************************
                                                                        ********
425     */
426     int
427     EDMRESubmitObj::deleteSubmitFiles()
428   1 {
429   1   // the iterator to traverse the tree of submit elements
430   1   RWBinaryTreeIterator *submitElementTree;
431   1   EDMRESubmitElement *elem;
432   1   char *submitfile;  // the name of the submit file to delete
433   1   int maxlen = MAXPATHLEN; // the max size of the submitfile
434   1   submitfile = (char *) calloc(1,maxlen);
                                  //calloc enough memory for the submit file
```



```
436   1   // if the submit file is null, we have a calloc failure
437   1   if (NULL == submitfile)
438   2     {
439   2       return -1;
440   1     }
442   1   // create an iterator to traverse the tree
443   1   submitElementTree = new RWBinaryTreeIterator(submitElements);
444   1   if (NULL == submitElementTree)
445   2     {
446   2       free (submitfile);
447   2       return (-1);
448   1     }
449   1   //reset the iterator
450   1   submitElementTree->reset();
451   1   // set to each submit element and get the
452   1   // submitfile name and unlink it
453   1   while(NULL != (*submitElementTree)())
454   2     {
455   2       elem = (EDMRESubmitElement *) submitElementTree->key();
456   2       if (NULL == elem)
457   3          {
458   3            free (submitfile);
459   3            return -1;
460   2          }
461   2       elem->getSubmitFile(submitfile, maxlen);
462   2       if (0 != strcmp(submitfile,"\0"))
463   3          {
464   3            unlink(submitfile);
465   2          }
466   1     }
468   1   //delete the submitElementTree which is the iterator
469   1   delete submitElementTree;
470   1   // free the calloced memory
471   1   free (submitfile);
472   1   return 0;
473     }
475     /************************************************************************
                                                                        ********
476     **
477     ** Routine:  getSubmitID
478     **
479     ** Inputs:   None
480     **
481     ** Outputs:  None
482     **
483     ** Return Codes:
484     **           int - the submit ID
485     **
486     ** Purpose:  Returns the submit ID.
487     **
488     ************************************************************************
                                                                        ********
489     */
491     int
492     EDMRESubmitObj::getSubmitID()
493   1 {
494   1     return submitID;
495     }
497     /************************************************************************
                                                                        ********
498     **
```

```
499  ** Routine:  setSubmitID
500  **
501  ** Inputs:   int submitid - ID of the submit job
502  **
503  ** Outputs:  None
504  **
505  ** Return Codes:
506  **           None
507  **
508  ** Purpose:  Sets the submit ID using the specified parameter.
509  **
510  ************************************************************************
                                                                      ********
511  */

513  void
514  EDMRESubmitObj::setSubmitID(int submitid)
515 1 {
516 1     submitID = submitid;
517  }

519  /************************************************************************
                                                                      ********
520  **
521  ** Routine:  getSubmitType
522  **
523  ** Inputs:   None
524  **
525  ** Outputs:  None
526  **
527  ** Return Codes:
528  **           unsigned int - the submit type
529  **
530  ** Purpose:  Returns the submit type.
531  **
532  ************************************************************************
                                                                      ********
533  */

535  unsigned int
536  EDMRESubmitObj::getSubmitType()
537 1 {
538 1     return so_submit_type;
539  }

541  /************************************************************************
                                                                      ********
542  **
543  ** Routine:  setSubmitType
544  **
545  ** Inputs:   unsigned int submittype - type of the submit job
546  **
547  ** Outputs:  None
548  **
549  ** Return Codes:
550  **           None
551  **
552  ** Purpose:  Sets the submit type using the specified parameter.
553  **
554  ************************************************************************
                                                                      ********
555  */

557  void
558  EDMRESubmitObj::setSubmitType(unsigned int submittype)
559 1 {
```



```
560 1     so_submit_type = submittype;
561  }

563  /************************************************************************
                                                                      ********
564  **
565  ** Routine:  getExecutionFlags
566  **
567  ** Inputs:   None
568  **
569  ** Outputs:  None
570  **
571  ** Return Codes:
572  **           unsigned int - the execution flags
573  **
574  ** Purpose:  Returns the execution flags.
575  **
576  ************************************************************************
                                                                      ********
577  */

579  unsigned int
580  EDMRESubmitObj::getExecutionFlags()
581 1 {
582 1     return so_execute_flags;
583  }

585  /************************************************************************
                                                                      ********
586  **
587  ** Routine:  setExecutionFlags
588  **
589  ** Inputs:   unsigned int executionflags - execution flags of the
                                                                  submit job
590  **
591  ** Outputs:  None
592  **
593  ** Return Codes:
594  **           None
595  **
596  ** Purpose:  Sets the execution flags using the specified parameter.
597  **
598  ************************************************************************
                                                                      ********
599  */

601  void
602  EDMRESubmitObj::setExecutionFlags(unsigned int executionflags)
603 1 {
604 1     so_execute_flags = executionflags;
605  }

607  /************************************************************************
                                                                      ********
608  **
609  ** Routine:  getNoVMCheck
610  **
611  ** Inputs:   None
612  **
613  ** Outputs:  None
614  **
615  ** Return Codes:
616  **           boolean_ty - TRUE if no VM check
617  **
618  ** Purpose:  Returns the VM check variable.
619  **
```

```
620   ************************************************************************
                                                                        ********
621   */

623   boolean_ty
624   EDMRESubmitObj::getNoVMCheck()
625 1 {
626 1     return so_no_vm_check;
627   }

629   /************************************************************************
                                                                        ********
630   **
631   ** Routine:  setNoVMCheck
632   **
633   ** Inputs:   boolean_ty novmcheck - no vm check boolean
634   **
635   ** Outputs:  None
636   **
637   ** Return Codes:
638   **           None
639   **
640   ** Purpose:  Sets the No-Vm-Check boolean to the specified parameter.
641   **
642   ************************************************************************
                                                                        ********
643   */

645   void
646   EDMRESubmitObj::setNoVMCheck(boolean_ty novmcheck)
647 1 {
648 1     so_no_vm_check = novmcheck;
649   }

651   /************************************************************************
                                                                        ********
652   **
653   ** Routine:  getPrePhaseExecutable
654   **
655   ** Inputs:   int maxsize - max size to copy to buffer
656   **
657   ** Outputs:  char *executable - executable to run in the pre-phase
658   **
659   ** Return Codes:
660   **           None
661   **
662   ** Purpose:  Returns the executable to run in the pre-phase in the
663   **           specified buffer.
664   **
665   ************************************************************************
                                                                        ********
666   */

668   void
669   EDMRESubmitObj::getPrePhaseExecutable(char *executable, int maxsize)
670 1 {
671 1     if (executable != NULL && pre_phase_executable != NULL)
672 2     {
673 2         strncpy(executable, pre_phase_executable, maxsize);
674 2         executable[maxsize - 1] = 0;
675 1     }
676 1     else if (executable != NULL)
677 2     {
678 2         executable[0] = 0;
679 1     }
680   }
```



```
682   /************************************************************************
                                                                        ********
683   **
684   ** Routine:  setPrePhaseExecutable
685   **
686   ** Inputs:   char *executable - executable to run in the pre-phase
687   **
688   ** Outputs:  None
689   **
690   ** Return Codes:
691   **           None
692   **
693   ** Purpose:  Sets the executable to run during the pre-phase using
694   **           the specified parameter.
695   **
696   ************************************************************************
                                                                        ********
697   */

699   void
700   EDMRESubmitObj::setPrePhaseExecutable(char *executable)
701 1 {
702 1     if (executable != NULL)
703 2     {
704 2         if (pre_phase_executable != NULL)
705 2             free(pre_phase_executable);

707 2         pre_phase_executable = strdup(executable);
708 1     }
709   }

711   /************************************************************************
                                                                        ********
712   **
713   ** Routine:  getPrePhaseArgs
714   **
715   ** Inputs:   None
716   **
717   ** Outputs:  char ***args - args to use in the pre-phase
718   **
719   ** Return Codes:
720   **           None
721   **
722   ** Purpose:  Returns the args to use in the pre-phase in the
723   **           specified buffer.
724   **
725   ************************************************************************
                                                                        ********
726   */

728   void
729   EDMRESubmitObj::getPrePhaseArgs(char ***args)
730 1 {
731 1 // Should make real copies of args and env
732 1     if (args != NULL)
733 1         *args = pre_phase_argv;
734   }

736   /************************************************************************
                                                                        ********
737   **
738   ** Routine:  setPrePhaseArgs
739   **
740   ** Inputs:   char **args - args to use in the pre-phase
741   **
```

```
742     ** Outputs:  None
743     **
744     ** Return Codes:
745     **           None
746     **
747     ** Purpose:  Sets the args to use during the pre-phase using
748     **           the specified parameter.
749     **
750     ************************************************************************
                                                                        ********
751     */

753     void
754     EDMRESubmitObj::setPrePhaseArgs(char **args)
755   1 {
756   1 // Should make real copies of args and env
757   1     if (args != NULL)
758   1         pre_phase_argv = args;
759     }

761     /************************************************************************
                                                                        ********
762     **
763     ** Routine:  getPrePhaseEnv
764     **
765     ** Inputs:   None
766     **
767     ** Outputs:  char ***env - env to use in the pre-phase
768     **
769     ** Return Codes:
770     **           None
771     **
772     ** Purpose:  Returns the env to use in the pre-phase in the
773     **           specified buffer.
774     **
775     ************************************************************************
                                                                        ********
776     */

778     void
779     EDMRESubmitObj::getPrePhaseEnv(char ***env)
780   1 {
781   1 // Should make real copies of args and env
782   1     if (env != NULL)
783   1         *env = pre_phase_env;
784     }

786     /************************************************************************
                                                                        ********
787     **
788     ** Routine:  setPrePhaseEnv
789     **
790     ** Inputs:   char **env - env to use in the pre-phase
791     **
792     ** Outputs:  None
793     **
794     ** Return Codes:
795     **           None
796     **
797     ** Purpose:  Sets the env to use during the pre-phase using
798     **           the specified parameter.
799     **
800     ************************************************************************
                                                                        ********
801     */
```



```
803     void
804     EDMRESubmitObj::setPrePhaseEnv(char **env)
805   1 {
806   1 // Should make real copies of args and env
807   1     if (env != NULL)
808   1         pre_phase_argv = env;
809     }

811     /************************************************************************
                                                                        ********
812     **
813     ** Routine:  getExecutePhaseExecutable
814     **
815     ** Inputs:   int maxsize - maximum size to copy into the executable
                                                                          buffer
816     **
817     ** Outputs:  char *executable - executable to run in the execute-phase
818     **
819     ** Return Codes:
820     **           None
821     **
822     ** Purpose:  Returns the executable to run in the execute-phase in the
823     **           specified buffer.
824     **
825     ************************************************************************
                                                                        ********
826     */

828     void
829     EDMRESubmitObj::getExecutePhaseExecutable(
                                                char *executable, int maxsize)
830   1 {
831   1     if (executable != NULL && execute_override_phase_executable !=
                                                                          NULL)
832   2     {
833   2         strncpy(
                      executable, execute_override_phase_executable, maxsize);
834   2         executable[maxsize - 1] = 0;
835   1     }
836   1     else if (executable != NULL)
837   2     {
838   2         executable[0] = 0;
839   1     }
840     }

842     /************************************************************************
                                                                        ********
843     **
844     ** Routine:  setExecutePhaseExecutable
845     **
846     ** Inputs:   char *executable - executable to run in the execute-phase
847     **
848     ** Outputs:  None
849     **
850     ** Return Codes:
851     **           None
852     **
853     ** Purpose:  Sets the executable to run during the execute-phase using
854     **           the specified parameter.
855     **
856     ************************************************************************
                                                                        ********
857     */

859     void
860     EDMRESubmitObj::setExecutePhaseExecutable(char *executable)
```

```
861 1 {
862 1     if (executable != NULL)
863 2     {
864 2         if (execute_override_phase_executable != NULL)
865 2             free(execute_override_phase_executable);

867 2         execute_override_phase_executable = strdup(executable);
868 1     }
869   }

871   /*************************************************************************
                                                                        ********

872   **
873   ** Routine:  getExecutePhaseArgs
874   **
875   ** Inputs:   None
876   **
877   ** Outputs:  char ***args - args to use in the execute-phase
878   **
879   ** Return Codes:
880   **           None
881   **
882   ** Purpose:  Returns the args to use in the execute-phase in the
883   **           specified buffer.
884   **
885   *************************************************************************
                                                                        ********

886   */

888   void
889   EDMRESubmitObj::getExecutePhaseArgs(char ***args)
890 1 {
891 1 // Should make real copies of args and env
892 1     if (args != NULL)
893 1         *args = execute_override_phase_argv;
894   }

896   /*************************************************************************
                                                                        ********

897   **
898   ** Routine:  setExecutePhaseArgs
899   **
900   ** Inputs:   char **args - args to use in the execute-phase
901   **
902   ** Outputs:  None
903   **
904   ** Return Codes:
905   **           None
906   **
907   ** Purpose:  Sets the args to use during the execute-phase using
908   **           the specified parameter.
909   **
910   *************************************************************************
                                                                        ********

911   */

913   void
914   EDMRESubmitObj::setExecutePhaseArgs(char **args)
915 1 {
916 1 // Should make real copies of args and env
917 1     if (args != NULL)
918 1         execute_override_phase_argv = args;
919   }

921   /*************************************************************************
                                                                        ********
```



```
922   **
923   ** Routine:  getExecutePhaseEnv
924   **
925   ** Inputs:   None
926   **
927   ** Outputs:  char ***env - env to use in the execute-phase
928   **
929   ** Return Codes:
930   **           None
931   **
932   ** Purpose:  Returns the env to use in the execute-phase in the
933   **           specified buffer.
934   **
935   *************************************************************************
                                                                        ********

936   */

938   void
939   EDMRESubmitObj::getExecutePhaseEnv(char ***env)
940 1 {
941 1 // Should make real copies of args and env
942 1     if (env != NULL)
943 1         *env = execute_override_phase_env;
944   }

946   /*************************************************************************
                                                                        ********

947   **
948   ** Routine:  setExecutePhaseEnv
949   **
950   ** Inputs:   char **env - env to use in the execute-phase
951   **
952   ** Outputs:  None
953   **
954   ** Return Codes:
955   **           None
956   **
957   ** Purpose:  Sets the env to use during the execute-phase using
958   **           the specified parameter.
959   **
960   *************************************************************************
                                                                        ********

961   */

963   void
964   EDMRESubmitObj::setExecutePhaseEnv(char **env)
965 1 {
966 1 // Should make real copies of args and env
967 1     if (env != NULL)
968 1         execute_override_phase_env = env;
969   }

971   /*************************************************************************
                                                                        ********

972   **
973   ** Routine:  getPostPhaseExecutable
974   **
975   ** Inputs:   int maxsize - maximum size to copy into the executable
                                                                          buffer
976   **
977   ** Outputs:  char *executable - executable to run in the post-phase
978   **
979   ** Return Codes:
980   **           None
981   **
982   ** Purpose:  Returns the executable to run in the post-phase in the
```

```
983     **           specified buffer.
984     **
985     ************************************************************************
                                                                        ********
986     */

988     void
989     EDMRESubmitObj::getPostPhaseExecutable(char *executable, int maxsize)
990   1 {
991   1     if (executable != NULL && post_phase_executable != NULL)
992   2     {
993   2         strncpy(executable, post_phase_executable, maxsize);
994   2         executable[maxsize - 1] = 0;
995   1     }
996   1     else if (executable != NULL)
997   2     {
998   2         executable[0] = 0;
999   1     }
1000  }

1002    /************************************************************************
                                                                        ********
1003    **
1004    ** Routine:  setPostPhaseExecutable
1005    **
1006    ** Inputs:   char *executable - executable to run in the post-phase
1007    **
1008    ** Outputs:  None
1009    **
1010    ** Return Codes:
1011    **           None
1012    **
1013    ** Purpose:  Sets the executable to run during the post-phase using
1014    **           the specified parameter.
1015    **
1016    ************************************************************************
                                                                        ********
1017    */

1019    void
1020    EDMRESubmitObj::setPostPhaseExecutable(char *executable)
1021  1 {
1022  1     if (executable != NULL)
1023  2     {
1024  2         if (post_phase_executable != NULL)
1025  2             free(post_phase_executable);

1027  2         post_phase_executable = strdup(executable);
1028  1     }
1029    }

1031    /************************************************************************
                                                                        ********
1032    **
1033    ** Routine:  getPostPhaseArgs
1034    **
1035    ** Inputs:   None
1036    **
1037    ** Outputs:  char ***args - args to use in the post-phase
1038    **
1039    ** Return Codes:
1040    **           None
1041    **
1042    ** Purpose:  Returns the args to use in the post-phase in the
1043    **           specified buffer.
1044    **
```



```
1045    ************************************************************************
                                                                        ********
1046    */

1048    void
1049    EDMRESubmitObj::getPostPhaseArgs(char ***args)
1050  1 {
1051  1 // Should make real copies of args and env
1052  1     if (args != NULL)
1053  1         *args = post_phase_argv;
1054    }

1056    /************************************************************************
                                                                        ********
1057    **
1058    ** Routine:  setPostPhaseArgs
1059    **
1060    ** Inputs:   char **args - args to use in the post-phase
1061    **
1062    ** Outputs:  None
1063    **
1064    ** Return Codes:
1065    **           None
1066    **
1067    ** Purpose:  Sets the args to use during the post-phase using
1068    **           the specified parameter.
1069    **
1070    ************************************************************************
                                                                        ********
1071    */

1073    void
1074    EDMRESubmitObj::setPostPhaseArgs(char **args)
1075  1 {
1076  1 // Should make real copies of args and env
1077  1     if (args != NULL)
1078  1         post_phase_argv = args;
1079    }

1081    /************************************************************************
                                                                        ********
1082    **
1083    ** Routine:  getPostPhaseEnv
1084    **
1085    ** Inputs:   None
1086    **
1087    ** Outputs:  char ***env - env to use in the post-phase
1088    **
1089    ** Return Codes:
1090    **           None
1091    **
1092    ** Purpose:  Returns the env to use in the post-phase in the
1093    **           specified buffer.
1094    **
1095    ************************************************************************
                                                                        ********
1096    */

1098    void
1099    EDMRESubmitObj::getPostPhaseEnv(char ***env)
1100  1 {
1101  1 // Should make real copies of args and env
1102  1     if (env != NULL)
1103  1         *env = post_phase_env;
1104    }
```

```
1106  /*************************************************************************
                                                                        ********
1107  **
1108  ** Routine:  setPostPhaseEnv
1109  **
1110  ** Inputs:   char **env - env to use in the post-phase
1111  **
1112  ** Outputs:  None
1113  **
1114  ** Return Codes:
1115  **           None
1116  **
1117  ** Purpose:  Sets the env to use during the post-phase using
1118  **           the specified parameter.
1119  **
1120  *************************************************************************
                                                                        ********
1121  */

1123  void
1124  EDMRESubmitObj::setPostPhaseEnv(char **env)
1125 1 {
1126 1 // Should make real copies of args and env
1127 1     if (env != NULL)
1128 1         post_phase_env = env;
1129  }

1131  /*************************************************************************
                                                                        ********
1132  **
1133  ** Routine:  IsBackupAdmin
1134  **
1135  ** Inputs:   None
1136  **
1137  ** Outputs:  None
1138  **
1139  ** Return Codes:
1140  **           boolean_ty - TRUE if the user is backup admin
1141  **
1142  ** Purpose:  Returns the backup admin boolean.
1143  **
1144  *************************************************************************
                                                                        ********
1145  */

1147  boolean_ty
1148  EDMRESubmitObj::IsBackupAdmin()
1149 1 {
1150 1     return so_backup_administrator;
1151  }

1153  /*************************************************************************
                                                                        ********
1154  **
1155  ** Routine:  setIsBackupAdmin
1156  **
1157  ** Inputs:   boolean_ty isadmin - is backup admin
1158  **
1159  ** Outputs:  None
1160  **
1161  ** Return Codes:
1162  **           None
1163  **
1164  ** Purpose:  Sets the backup admin boolean to the specified parameter.
1165  **
1166  *************************************************************************
```



```
                                                                        ********
1167  */

1169  void
1170  EDMRESubmitObj::setIsBackupAdmin(boolean_ty isadmin)
1171 1 {
1172 1     so_backup_administrator = isadmin;
1173  }

1175  /*************************************************************************
                                                                        ********
1176  **
1177  ** Routine:  IsSourceSystemAdmin
1178  **
1179  ** Inputs:   None
1180  **
1181  ** Outputs:  None
1182  **
1183  ** Return Codes:
1184  **           boolean_ty - TRUE if the user is admin on client system
1185  **
1186  ** Purpose:  Returns the backup admin boolean.
1187  **
1188  *************************************************************************
                                                                        ********
1189  */

1191  boolean_ty
1192  EDMRESubmitObj::IsSourceSystemAdmin()
1193 1 {
1194 1     return so_src_sys_admin;
1195  }

1197  /*************************************************************************
                                                                        ********
1198  **
1199  ** Routine:  setIsBackupAdmin
1200  **
1201  ** Inputs:   boolean_ty issrcadmin - is admin on client machine
1202  **
1203  ** Outputs:  None
1204  **
1205  ** Return Codes:
1206  **           None
1207  **
1208  ** Purpose:  Sets the admin-on-client boolean to the specified
                                                                    parameter.
1209  **
1210  *************************************************************************
                                                                        ********
1211  */

1213  void
1214  EDMRESubmitObj::setIsSourceSystemAdmin(boolean_ty issrcadmin)
1215 1 {
1216 1     so_src_sys_admin = issrcadmin;
1217  }

1219  /*************************************************************************
                                                                        ********
1220  **
1221  ** Routine:  IsDestinationAdmin
1222  **
1223  ** Inputs:   None
1224  **
1225  ** Outputs:  None
```

```
1226    **
1227    ** Return Codes:
1228    **            boolean_ty - TRUE if the user is destination machine
                                                                        admin
1229    **
1230    ** Purpose:  Returns the destination admin boolean.
1231    **
1232    ************************************************************************
                                                                        ********
1233    */

1235    boolean_ty
1236    EDMRESubmitObj::IsDestinationAdmin()
1237  1 {
1238  1     return so_dst_sys_admin;
1239    }

1241    /************************************************************************
                                                                        ********
1242    **
1243    ** Routine:  setIsDestinationAdmin
1244    **
1245    ** Inputs:   boolean_ty isdestadmin - is destination host admin
1246    **
1247    ** Outputs:  None
1248    **
1249    ** Return Codes:
1250    **            None
1251    **
1252    ** Purpose:  Sets the destination-host-admin boolean to the specified
                                                                      parameter.
1253    **
1254    ************************************************************************
                                                                        ********
1255    */

1257    void
1258    EDMRESubmitObj::setIsDestinationAdmin(boolean_ty isdestadmin)
1259  1 {
1260  1     so_dst_sys_admin = isdestadmin;
1261    }

1263    /************************************************************************
                                                                        ********
1264    **
1265    ** Routine:  getUserID
1266    **
1267    ** Inputs:   None
1268    **
1269    ** Outputs:  None
1270    **
1271    ** Return Codes:
1272    **            uid_t - the userid of the user
1273    **
1274    ** Purpose:  Returns the user ID of the user doing the restore.
1275    **
1276    ************************************************************************
                                                                        ********
1277    */

1279    uid_t
1280    EDMRESubmitObj::getUserID()
1281  1 {
1282  1     return so_human_uid;
1283    }
```



```
1285    /************************************************************************
                                                                        ********
1286    **
1287    ** Routine:  setUserID
1288    **
1289    ** Inputs:   uid_t uid - userid of user
1290    **
1291    ** Outputs:  None
1292    **
1293    ** Return Codes:
1294    **            None
1295    **
1296    ** Purpose:  Sets the userid to the specified parameter.
1297    **
1298    ************************************************************************
                                                                        ********
1299    */

1301    void
1302    EDMRESubmitObj::setUserID(uid_t uid)
1303  1 {
1304  1     so_human_uid = uid;
1305    }

1307    /************************************************************************
                                                                        ********
1308    **
1309    ** Routine:  getEffectiveUserID
1310    **
1311    ** Inputs:   None
1312    **
1313    ** Outputs:  None
1314    **
1315    ** Return Codes:
1316    **            uid_t - the effective userid of the user
1317    **
1318    ** Purpose:  Returns the effective user ID of the user doing the
                                                                      restore.
1319    **
1320    ************************************************************************
                                                                        ********
1321    */

1323    uid_t
1324    EDMRESubmitObj::getEffectiveUserID()
1325  1 {
1326  1     return so_effective_uid;
1327    }

1329    /************************************************************************
                                                                        ********
1330    **
1331    ** Routine:  setEffectiveUserID
1332    **
1333    ** Inputs:   uid_t uid - effective userid of user
1334    **
1335    ** Outputs:  None
1336    **
1337    ** Return Codes:
1338    **            None
1339    **
1340    ** Purpose:  Sets the effective userid to the specified parameter.
1341    **
1342    ************************************************************************
                                                                        ********
1343    */
```

```
1345    void
1346    EDMRESubmitObj::setEffectiveUserID(uid_t uid)
1347  1 {
1348  1     so_effective_uid = uid;
1349    }

1351    /*************************************************************************
                                                                        ********

1352    **
1353    ** Routine:  getUserName
1354    **
1355    ** Inputs:   int maxsize - size of the buffer passed in
1356    **
1357    ** Outputs:  char *username - name of the user
1358    **
1359    ** Return Codes:
1360    **           None
1361    **
1362    ** Purpose:  Returns the name of the user in the specified buffer.
1363    **
1364    *************************************************************************
                                                                        ********

1365    */

1367    void
1368    EDMRESubmitObj::getUserName(char *username, int maxsize)
1369  1 {
1370  1     if (username != NULL && so_human_uidname != NULL)
1371  2     {
1372  2         strncpy(username, so_human_uidname, maxsize);
1373  2         username[maxsize - 1] = 0;
1374  1     }
1375  1     else if (username != NULL)
1376  2     {
1377  2         username[0] = 0;
1378  1     }
1379    }

1381    /*************************************************************************
                                                                        ********

1382    **
1383    ** Routine:  setUserName
1384    **
1385    ** Inputs:   char *username - name of the user
1386    **
1387    ** Outputs:  None
1388    **
1389    ** Return Codes:
1390    **           None
1391    **
1392    ** Purpose:  Sets the name of the user in the specified parameter.
1393    **
1394    *************************************************************************
                                                                        ********

1395    */

1397    void
1398    EDMRESubmitObj::setUserName(char *username)
1399  1 {
1400  1     if (username != NULL)
1401  2     {
1402  2         if (so_human_uidname != NULL)
1403  2             free(so_human_uidname);

1405  2         so_human_uidname = strdup(username);
```



```
1406  1     }
1407    }

1409    /*************************************************************************
                                                                        ********

1410    **
1411    ** Routine:  getEffectiveUserName
1412    **
1413    ** Inputs:   int maxsize - size of the buffer passed in
1414    **
1415    ** Outputs:  char *username - name of the user
1416    **
1417    ** Return Codes:
1418    **           None
1419    **
1420    ** Purpose:  Returns the name of the user in the specified buffer.
1421    **
1422    *************************************************************************
                                                                        ********

1423    */

1425    void
1426    EDMRESubmitObj::getEffectiveUserName(char *username, int maxsize)
1427  1 {
1428  1     if (username != NULL && so_effective_uidname != NULL)
1429  2     {
1430  2         strncpy(username, so_effective_uidname, maxsize);
1431  2         username[maxsize - 1] = 0;
1432  1     }
1433  1     else if (username != NULL)
1434  2     {
1435  2         username[0] = 0;
1436  1     }
1437    }

1439    /*************************************************************************
                                                                        ********

1440    **
1441    ** Routine:  setEffectiveUserName
1442    **
1443    ** Inputs:   char *username - name of the user
1444    **
1445    ** Outputs:  None
1446    **
1447    ** Return Codes:
1448    **           None
1449    **
1450    ** Purpose:  Sets the name of the user in the specified parameter.
1451    **
1452    *************************************************************************
                                                                        ********

1453    */

1455    void
1456    EDMRESubmitObj::setEffectiveUserName(char *username)
1457  1 {
1458  1     if (username != NULL)
1459  2     {
1460  2         if (so_effective_uidname != NULL)
1461  2             free(so_effective_uidname);

1463  2         so_effective_uidname = strdup(username);
1464  1     }
1465    }

1467    /*************************************************************************
```

```
                                                                        ********
1468    **
1469    ** Routine:  getMarkSummary
1470    **
1471    ** Inputs:   None
1472    **
1473    ** Outputs:  struct mark_summary *markptr - place to put mark summary
1474    **
1475    ** Return Codes:
1476    **           None
1477    **
1478    ** Purpose:  Returns the mark summary in the specified buffer.
1479    **
1480    ***************************************************************************
                                                                        ********
1481    */

1483    void
1484    EDMRESubmitObj::getMarkSummary(struct mark_summary *markptr)
1485  1 {
1486  1     if (markptr != NULL)
1487  1         memcpy(markptr, &so_total_submit_summary, sizeof(
                                                    struct mark_summary));
1488    }

1490    /***************************************************************************
                                                                        ********
1491    **
1492    ** Routine:  setMarkSummary
1493    **
1494    ** Inputs:   struct mark_summary *markptr - place to put the mark
                                                                          summary
1495    **
1496    ** Outputs:  None
1497    **
1498    ** Return Codes:
1499    **           None
1500    **
1501    ** Purpose:  Sets the mark summary using the specified buffer.
1502    **
1503    ***************************************************************************
                                                                        ********
1504    */

1506    void
1507    EDMRESubmitObj::setMarkSummary(struct mark_summary *markptr)
1508  1 {
1509  1     if (markptr != NULL)
1510  1         memcpy(&so_total_submit_summary, markptr, sizeof(
                                                    struct mark_summary));
1511    }

1513    /***************************************************************************
                                                                        ********
1514    **
1515    ** Routine:  getVolumeList
1516    **
1517    ** Inputs:   None
1518    **
1519    ** Outputs:  ebvl_volidlist_ty **volptr - place to put the volume list
1520    **
1521    ** Return Codes:
1522    **           None
1523    **
1524    ** Purpose:  Returns the volume list in the specified buffer.
1525    **
```



```
1526    ***************************************************************************
                                                                        ********
1527    */

1529    void
1530    EDMRESubmitObj::getVolumeList(ebvl_volidlist_ty **volptr)
1531  1 {
1532  1     if (volptr != NULL)
1533  1         *volptr = so_total_volume_list;
1534    }

1536    /***************************************************************************
                                                                        ********
1537    **
1538    ** Routine:  setVolumeList
1539    **
1540    ** Inputs:   ebvl_volidlist_ty *volptr - place to put the volume list
1541    **
1542    ** Outputs:  None
1543    **
1544    ** Return Codes:
1545    **           None
1546    **
1547    ** Purpose:  Sets the volume list using the specified buffer.
1548    **
1549    ***************************************************************************
                                                                        ********
1550    */

1552    void
1553    EDMRESubmitObj::setVolumeList(ebvl_volidlist_ty *volptr)
1554  1 {
1555  1     if (volptr != NULL)
1556  1         so_total_volume_list = volptr;
1557    }

1559    /***************************************************************************
                                                                        ********
1560    **
1561    ** Routine:  newSubmitElement
1562    **
1563    ** Inputs:   None
1564    **
1565    ** Outputs:  None
1566    **
1567    ** Return Codes:
1568    **           int - the ID of the Submit Element
1569    **
1570    ** Purpose:  Creates a new SubmitElement object and returns the ID.
1571    **
1572    ***************************************************************************
                                                                        ********
1573    */

1575    int
1576    EDMRESubmitObj::newSubmitElement()
1577  1 {
1578  1       EDMRESubmitElement *se;
1579  1       EDMRESubmitElement *ret;

1581  1       se = new EDMRESubmitElement();

1583  1       if (se == NULL)
1584  1           return -1;

1586  1       se -> setSubmitElementID(++so_witem_count);
```

```
1588 1      ret = (EDMRESubmitElement *) submitElements.insert(se);

1590 1      if (ret == NULL)
1591 2      {
1592 2          delete se;
1593 2          so_witem_count--;
1594 2          return -1;
1595 1      }

1597 1      return so_witem_count;
1598    }

1600    /*************************************************************************
                                                                        ********
1601    **
1602    ** Routine:  getSubmitElement
1603    **
1604    ** Inputs:   int submitElementID - ID to look up and return
1605    **
1606    ** Outputs:  None
1607    **
1608    ** Return Codes:
1609    **           EDMRESubmitElement - return the SubmitElement based on
                                                                        the ID
1610    **
1611    ** Purpose:  Looks up and returns a SubmitElement based on the Element
                                                                        ID
1612    **
1613    *************************************************************************
                                                                        ********
1614    */

1616    EDMRESubmitElement *
1617    EDMRESubmitObj::getSubmitElement(int submitElementID)
1618 1  {
1619 1      EDMRESubmitElement *se;
1620 1      EDMRESubmitElement *ret;

1622 1      se = new EDMRESubmitElement();

1624 1      if (se == NULL)
1625 1          return NULL;

1627 1      se -> setSubmitElementID(submitElementID);

1629 1      ret = (EDMRESubmitElement *) submitElements.find(se);

1631 1      delete se;

1633 1      return ret;
1634    }

1636    /*************************************************************************
                                                                        ********
1637    **
1638    ** Routine:  removeSubmitElement
1639    **
1640    ** Inputs:   int submitElementID - ID to look up and remove
1641    **
1642    ** Outputs:  None
1643    **
1644    ** Return Codes:
1645    **           None
1646    **
1647    ** Purpose:  Looks up and removes a SubmitElement based on the Element
```



```
                                                                        ID
1648    **
1649    *************************************************************************
                                                                        ********
1650    */

1652    void
1653    EDMRESubmitObj::removeSubmitElement(int submitElementID)
1654 1  {
1655 1      EDMRESubmitElement *se;
1656 1      EDMRESubmitElement *ret;

1658 1      se = new EDMRESubmitElement();

1660 1      if (se == NULL)
1661 1          return;

1663 1      se -> setSubmitElementID(submitElementID);

1665 1      ret = (EDMRESubmitElement *) submitElements.find(se);

1667 1      if (ret == NULL)
1668 2      {
1669 2          delete se;
1670 2          return;
1671 1      }

1673 1      ret = (EDMRESubmitElement *) submitElements.remove(ret);

1675 1      if (ret == NULL)
1676 2      {
1677 2          delete se;
1678 2          return;
1679 1      }

1681 1      delete se;
1682 1      delete ret;
1683    }

1685    /*************************************************************************
                                                                        ********
1686    **
1687    ** Routine:  getWICount
1688    **
1689    ** Inputs:   None
1690    **
1691    ** Outputs:  None
1692    **
1693    ** Return Codes:
1694    **           int - the number of submit elements
1695    **
1696    ** Purpose:  Returns the number of submit elements.
1697    **
1698    *************************************************************************
                                                                        ********
1699    */

1701    int
1702    EDMRESubmitObj::getWICount()
1703 1  {
1704 1      return so_witem_count;
1705    }
```